МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни
"Грід обчислення та хмарні технології"

для студентів спеціальності 123 "Комп'ютерна інженерія"
всіх форм навчання

2018

Методичні вказівки до лабораторних робіт з дисципліни "Грід обчислення та хмарні технології" для студентів спеціальності 123 "Комп'ютерна інженерія" всіх форм навчання / Укл. С.Ю. Скрупський, В.В. Шкарупило – Запоріжжя: ЗНТУ, 2018. – 64 с.

Укладачі: С.Ю. Скрупський, к.т.н., доцент
В.В. Шкарупило, к.т.н., доцент

Рецензент: Г.Г. Киричек, к.т.н., доцент

Відповідальний за випуск: С.Ю. Скрупський, к.т.н., доцент

Затверджено:
на засіданні кафедри
"Комп’ютерні системи та мережі"
Протокол № 7 від 21.03.18

Рекомендовано до видання
НМК ФКНТ
Протокол № 8 від 27.03.2018

ЗМІСТ

ЛАБОРАТОРНА РОБОТА № 1

Створення віртуальної організації *Grid*-кластера

Мета роботи – навчитися створювати та конфігурувати віртуальну організацію *Grid*-кластера.

1.1 Теоретичні відомості

Віртуальна організація (*VO*) – основа організаційної структури *Grid* – об'єднання підприємств, установ та окремих спеціалістів, що мають спільну мету та використовують спільні ресурси. Ресурси можуть бути представлені, зокрема, у вигляді кластерів.

VO Management Service (*VOMS*) – пакет інструментарію *gLite*, призначений для підтримки та обслуговування користувачів *VO*. *VOMS* є централізованим репозитарієм для збереження аутентифікаційних даних користувачів віртуальних організацій. Цей сервіс дозволяє об'єднувати користувачів у групи, наділяючи ці групи певними пріоритетами. Забезпечується також можливість вести статистику активності користувачів тощо.

Ключові компоненти *VOMS* наведено на рис. 1.1.

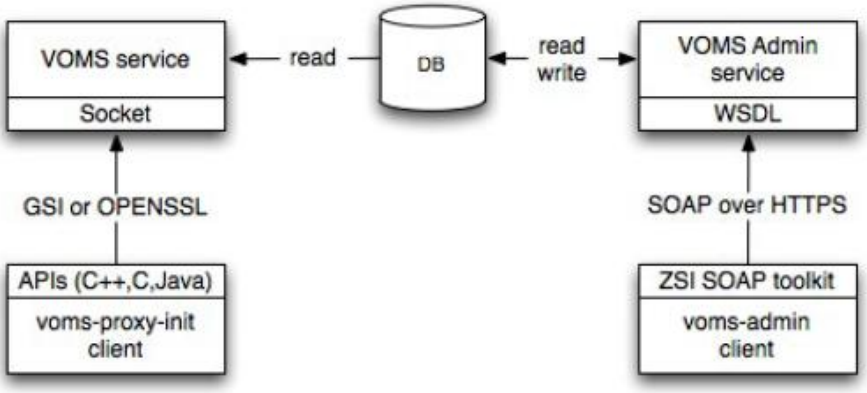


Рисунок 1.1 – Архітектура *VOMS*

В якості бази даних (*DB* на схемі) будемо використовувати MySQL.

VOMS Admin – це веб-додаток, що підтримує засоби адміністрування баз даних, яким відповідають певні ВО. VOMS Admin реалізує інтуїтивний інтерфейс користувача, для вирішення щоденних задач адміністрування. Для забезпечення віддаленого адміністрування, використовується інтерфейс SOAP.

Захист даних, що передаються між компонентами gLite, впроваджується також за рахунок сертифікатів, які будуть згенеровані за допомогою пакету OpenSSL.

OpenSSL – криптографічний пакет, з відкритим вихідним кодом. Цей пакет є інтегрованим в SL4.8. OpenSSL призначений для роботи з SSL/TLS. Він, зокрема, дозволяє створювати ключі RSA та сертифікати x.509.

Grid Security Infrastructure (GSI) – специфікація, що підтримує криптографічно захищену взаємодію програмних компонентів та обчислювального середовища Grid, при цьому забезпечується підтримка механізму делегування прав. Для підтримки цих можливостей використовується асиметричне шифрування.

1.2 Підготовчий етап

1.2.1 У меню вибору завантаження операційної системи вибрати **Scientific Linux 4.8**.

1.2.2 Ввести логін **root**, пароль – **root.**

1.2.3 Змінити ім’я хосту на таке, що відповідає номеру вашого комп’ютера: **System Settings → Network**. На вкладці **DNS** у полі **Hostname** (рис. 1.2) вказати ім’я хосту, що відповідає наступному специфікатору: **voms*X***, де ***X* –** порядковий номер комп’ютера.

1.2.4 Змінити ip-адресу хосту у відповідності до номеру ПК: на вкладці **Devices** натиснути **Edit** та ввести у відповідних полях статичну ip-адресу **10.0.32.*X*** з маскою **255.255.255.0,** шлюз **10.0.32.50**, де ***X* –** порядковий номер комп’ютера.

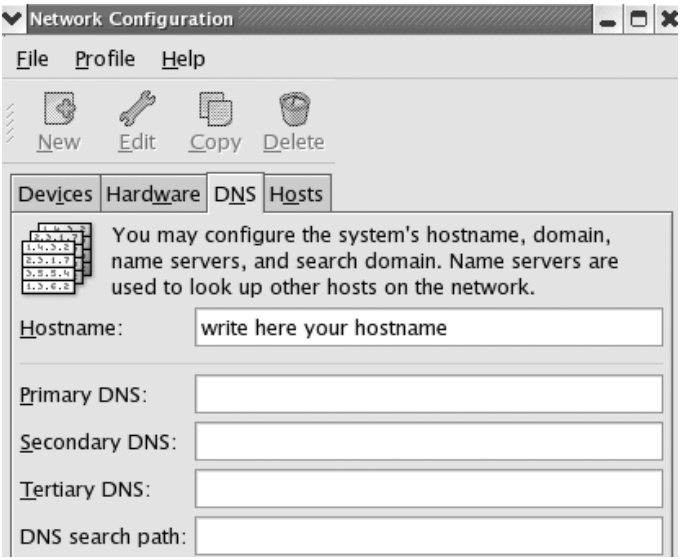


Рисунок 1.2 – Зміна доменного імені хосту

1.3 Порядок виконання роботи

1.3.1 **БД MySQL**

Запустити MySQL: > /etc/init.d/mysqld start. Перейти в інтерпретатор (CLI) командного рядка MySQL: > mysql -u root –p. На запит «Enter password:» ввести «**root**». Створити БД, з якою буде асоціюватися ваша віртуальна організація. В якості **db_name** вказати будь-яку назву: mysql> create database <db_name>;

Забезпечити доступ користувачів ВО до цієї БД: mysql> GRANT ALL PRIVILEGES ON <db_name>.* TO 'root'@'localhost' \
IDENTIFIED BY 'root';

Вийти з CLI: mysql> \q

1.3.2 **Tomcat** використовується в якості самостійного веб-серверу (для доступу користувачів до ВО). Запустити Tomcat: > /etc/init.d/tomcat5 start

1.3.3 **OPENSSL**

Згенеруємо 2 ключі RSA та сертифікат x.509: hostkey.pem – закритий ключ, hostcert.pem – відкритий ключ.

> cd /etc/grid-security/sslcert

> openssl req \

> -x509 -nodes -days 365 \

> -newkey rsa:1024 -keyout hostkey.pem -out hostcert.pem -config ./openssl.cnf

Ці рядки свідчать про генерацію двох ключів, сумісніх з протоколом **-x509**, термін дії цих ключів складає **365** днів, кожний ключ складається з **1024** байтів, для створення форми використовується конфігураційний файл **openssl.cnf**, що знаходиться у поточній директорії.

Заповнити запропоновану форму, приклад якої наведено в лістингу 1.3.

Лістинг 1.3 – Форма для введення даних про ВО

```
-----------------------------------------------------------------------------
Organization Name (company) [My Company]:                    zntu
Organizational Unit Name (department, division) []:          ksm
Email Address []:                                            vadshkar@yandex.ru
Locality Name (city, district) [My Town]:                    Zaporozhye
State or Province Name (full name) [State or Providence]:    state
Country Name (2 letter code) [US]:                           UA
Common Name (hostname, IP, or your name) []:                 localhost
-----------------------------------------------------------------------------
```

Введені дані будуть використовуватися для генерування двох вищеназваних RSA-ключів.

Скопіювати згенеровані ключі у директорію /etc/grid-security.

Змінити права доступу до ключів:

> cd /etc/grid-security

> chmod 0644 hostcert.pem # доступний публічно

> chmod 0400 hostkey.pem # закритий ключ (лише для адміністратора)

> cd ..

> chmod 0400 grid-security # закрита директорія

1.3.4 **Конфігураційний файл vo-list.cfg.xml**

Модифікувати конфігураційний файл vo-list.cfg.xml, що використовується для збереження конфігураційної інформації про відомі ВО. Файл розміщений за наступним шляхом:

> cd /opt/glite/etc/config/

Необхідно внести зміни в наступні параметри:

- "vo_name" - вказати "назву_вашої_ВО";

- "port_number" - вказати число (воно має бути більше за 15000), "port_number" вказує на номер порту, на якому VOMS буде прослуховувати вхідні запити від ВО. Зазвичай, обирають номер порту у межах: 15001…15100.
- "db_name" - вказати назву створеної вами бази даних MySQL, яка буде використовуватися для збереження записів про користувачів ВО.

1.3.5 **Перевірка конфігурації**

Ввести в bash наступні рядки:

> cd /opt/glite/etc/config

> scripts/glite-voms-server-config.py --checkconf

Якщо результат позитивний та схожий на той, що наведений на рис. 1.4, можна переходити до виконання наступних пунктів.

1.3.6 **Конфігурування ВО**

> scripts/glite-voms-server-config.py --configure

1.3.7 **Запуск ВО**

> scripts/glite-voms-server-config.py --start

1.4 Зміст звіту

1.4.1 Титульний лист.

1.4.2 Мета роботи.

1.4.3 Відомості про створену віртуальну організацію як наведено в лістингу 1.4.

1.4.4 Відповіді на контрольні питання.

1.5 Контрольні питання

1.5.1 Що таке віртуальна організація та мета її створення?

1.5.2 Призначення VOMS.

1.5.3 Навести та пояснити архітектуру VOMS.

1.5.4 Описати компонент VOMS Admin.

1.5.5 Призначення та параметри файлу vo-list.cfg.xml.

1.5.6 Призначення та можливості пакету OpenSSL.

1.5.7 Призначення GSI.

Лістинг 1.4 – Приклад правильно створеної ВО

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            Main settings of gLite VOMS Server
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

  VOMS SERVER
     ------------------------------------------------------------
     [DB type]             mysql
     [VOMS-admin enabled] true
     [VOMS endpoint]       https://slinux:8443/vomses/
     ------------------------------------------------------------

  VOMS VO settings
     ------------------------------------------------------------

     [VO] vadim_vo
         ============================================================
         [VOMS VO endpoint]          https://slinux:8443/voms/vadim_vo
         [VOMS Hostname]             localhost
         [VOMS Port]                 15001
         [VOMS Proxy Timeout]        86400s
         [VOMS Short FQANs]          false
         [VOMS logrotation period]  daily
         [VOMS logrotation number]  90
         ------------------------------------------------------
         [VOMS-admin SMTP]     localhost
         [VOMS-admin mail]     vadshkar@yandex.ru
         [VOMS-admin cert]
         [VOMS-admin disable web registration] false
         [VOMS-admin membership request timeout] 86400s
         [VOMS-admin e-mail user when membership request expires] true
         [VOMS-admin SAML max assertion Lifetime] 720s
         ------------------------------------------------------
         [DB Name]             vadim_vo_db
         [DB UserName]         root
         [DB UserPassword]     vadim
         [DB Host]             localhost
         [DB AdminName]        root
         [DB AdminPassword]    vadim
         [DB Port]             3306
         ============================================================
```

ЛАБОРАТОРНА РОБОТА № 2

Конфігурування обчислювального та керуючого вузлів GRID – кластера, побудованого на базі middleware gLite 3.1

Мета роботи – вивчити складові компоненти gLite 3.1 та їх призначення, набути практичних навичок конфігурування обчислювальних вузлів з керуючим (сайтом віртуальної організації).

2.1 Теоретичні відомості

GRID – географічно розподілена інфраструктура, що об'єднує ресурси різного типу (процесори, оперативна пам'ять, тощо). Основою концепції GRID є колективний розподілений режим доступу до ресурсів та пов'язаних з ними сервісів, в рамках розподілених віртуальних організацій.

GLite 3.1 - новітнє покоління middleware для систем GRID. Крім інструментальних засобів до нього входить широкий набір служб.

Компоненти gLite 3.1, що будуть задіяні в лабораторній роботі, наведені нижче:

- *Worker Node* (**WN**) – вузол, який безпосередньо виконує обчислення;
- *Computing Element* (**CE**) – набір програм, що встановлюється на керуючий вузол обчислювального кластера, являє собою універсальний інтерфейс до системи керування ресурсами кластера та дозволяє запускати обчислювальні завдання;
- *TORQUE-server, TORQUE-utils* – серверна частина менеджера локальних ресурсів, що встановлюються на керуючому вузлі – менеджер локальних ресурсів відповідає за розподіл обчислень між **WN** кластера;
- *TORQUE-client* – компонент менеджера локальних ресурсів, що встановлюються на обчислювальному вузлі (**WN**) та виконує роль агента, що взаємодіє з серверною частиною, розміщеною на керуючому вузлі.
- *BDII* – інформаційний вказівник по базі даних Берклі (не реляційна високопродуктивна інтегрована база даних, що реалізована у вигляді бібліотеки).

В даній роботі буде виконуватися з'єднання та конфігурування керуючого та обчислювального вузлів. Прив'язка буде здійснюватися на основі імен хостів.

2.2 Підготовчий етап

2.2.1 У меню вибору завантаження операційної системи вибрати **Scientific Linux 4.8**.

2.2.2 Ввести логін **root**, пароль – **root**.

2.2.3 Змінити ім'я хосту на таке, що відповідає номеру вашого комп'ютера: **System Settings → Network**. На вкладці **DNS** у полі **Hostname** вказати ім'я хосту, що відповідає наступному специфікатору:

- **wn*X*** – якщо це обчислювальний вузол**;**
- **site** – якщо це керуючий вузол;
- ***X*** **–** порядковий номер комп'ютера.

1.2.5 Змінити ір-адресу хосту у відповідності до номеру ПК: на вкладці **Devices** натиснути **Edit** та ввести у відповідних полях статичну ір-адресу **10.0.32.*X*** з маскою **255.255.255.0,** шлюз **10.0.32.50**, де ***X*** **–** порядковий номер комп'ютера.

1.2.6 Відредагувати файл **/etc/hosts**, що створює локальну таблицю маршрутизації:

- відкрити термінал:

> **cd /etc**
> **mc** # викликати Midnight Commander

- відкрити файл **hosts,** виділивши його та натиснувши **F4**;
- не видаляти записи у файлі, а модифікувати 2 останні рядки як зображено на рис. 2.1.

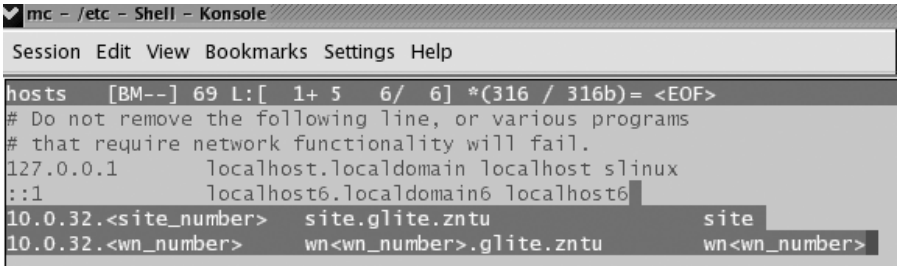


Рисунок 2.1 – Приклад модифікації файлу хостів

- <**site_number**> - номер сусіднього (власного) обчислювального вузла, де встановлений **site**;
- <**wn_number**> - номер власного (сусіднього) обчислювального вузла, де встановлений **wn*X***.

Наприклад, якщо Ви знаходитеся за керуючим вузлом, то, в якості <**site_number**> вводите номер власного робочого вузла, в якості <**wn_number**> вводите номер обчислювального вузла (сусіднього вузла). Якщо Ви знаходитеся за обчислювальним вузлом, то, в якості <**site_number**> вводите номер керуючого вузла сусіда, а в якості <**wn_number**> вводите номер власного вузла.

Таким чином, в результаті виконання цих дій, 2 останні рядки файлів **hosts** опонентів (обчислювального та керуючого вузлів) мають співпадати: це робиться для того, щоб керуючий та обчислювальний вузли «бачили» один одного (введені імена хостів будуть використовуватися при подальшому конфігуруванні).

Перезавантажити обидва комп'ютери.

Для перевірки правильності виконаних дій, ввести в терміналі наступний рядок:

запустити службу **ssh**, для обміну даними між хостами:

> **/etc/init.d/sshd start**

На обох вузлах (обчислювальному та керуючому) служба **ssh** має бути запущеною.

> **ssh <login>@<hostname>**

наприклад:

> **ssh root@wn19**

намагаємося підключитися до обчислювального вузла № 19.

Перевірити, чи синхронізовані керуючий та обчислювальний вузли:

> service ntpd status

Якщо все виконане правильно, Вам буде запропоновано під'єднатися до вузла, ввівши пароль (**root**).

2.3 Порядок виконання роботи

Встановлення компонентів виконується з офіційних репозиторіїв, що доступні на сайті cern: http://glite.web.cern.ch/glite/packages/R3.1/. Для керуючого вузла потрібні наступні компоненти:

- TORQUE-server;
- TORQUE-utils;
- glite-BDII;
- LCG-CE.

Для обчислювального вузла потрібні такі компоненти:

- glite-WN;
- TORQUE-client.

Після збереження репозиторіїв на комп'ютері, встановлення компонентів виконується за допомогою команди **yum install <список_компонентів>**. Оскільки загальний розмір необхідних компонентів складає сотні мегабайт, а їх інсталяція займає багато часу, для лабораторної роботи ці компоненти вже встановлені.

Конфігурування вузлів. **GLite** встановлюється в директорію **/opt**: (**/opt/glite**). Для конфігурування компонентів **gLite 3.1** створено спеціальний менеджер – **yaim**. Директорія **yaim** знаходиться за наступним шляхом: **/opt/glite/yaim**. Вихідні конфігураційні файли розміщені в директорії **examples**: **/opt/glite/yaim/examples.** Ці файли необхідно буде модифікувати: для цього потрібно скопіювати файли **site-info.def, users.conf, groups.conf** та **wn-list.conf** у директорію **/opt/glite/yaim/etc.** Відповідні пояснення щодо синтаксису конфігураційних файлів знаходяться у директорії **/opt/glite/yaim/examples**, у файлах з аналогічними назвами, але з розширеннями ***.README**.

Усі модифікації потрібно виконувати над скопійованими файлами, у директорії **/opt/glite/yaim/etc**. Конфігураційні файли мають бути однаковими і на обчислювальному, і на керуючому вузлах. Вам необхідно буде змінити копії цих файлів так, як вказано нижче.

Опис конфігураційних файлів:

- **site-info.def** – містить більшість конфігураційних налаштувань (використовується при налаштуванні усіх вузлів **gLite**). Вміст файлу та опис його параметрів наводиться в лістингу 2.1. Необхідно модифікувати **/opt/glite/yaim/etc/site-info.def**. Екземпляр файлу **site-info.def,** який необхідно змінити, розміщений в директорії **/var/cache/yum/top-secret**.

- **users.conf** – містить список користувачів GRID заданої віртуальної організації, має виглядати наступним чином:

3001:zntu_vogrid001:1400:zntu_vo:zntu_vo::
3002:zntu_vogrid002:1400:zntu_vo:zntu_vo:sgm:

Тут **3001** – унікальний ідентифікатор окремого користувача заданої віртуальної організації **zntu_vo**, **zntu_vogrid001 (002)** – це логін окремого користувача **zntu_vo**, **1400** – унікальний ідентифікатор групи, до якої належать користувачі вказаної віртуальної організації.

Передостанній наведений ідентифікатор свідчить про те, що група користувачів має назву **zntu_vo**, а останній параметр записів вказує вже безпосередньо назву віртуальної організації, до якої входять користувачі заданої групи. Ранжування на групи дозволяє більш гнучко диференціювати користувачів за напрямами їх спрямувань (розмежовувати спектр дозволених можливостей та привілеїв).

- **groups.conf** – містить список груп користувачів заданої віртуальної організації (**zntu_vo**). Вміст цього файлу повинен мати наступний вигляд:

 # zntu_vo group
"/zntu_vo/ROLE=lcgadmin":::sgm:
"/zntu_vo"::::

Другий рядок цього запису свідчить про те, що адміністратор віртуальної організації **zntu_vo,** який має ідентифікатор **lcgadmin,** має права додавати до зазначеної віртуальної організації нових учасників, про що свідчить прапорець **sgm.**

- **wn-list.conf** – містить список імен хостів (доменних імен), про яких відомо **CE.** Має вказуватися один запис в рядку. Приклад файлу наведений нижче:

#wn-list.conf
<**wn_number**>

Тут <**wn_number**> - номер обчислювального вузла, де встановлений **wn*X*** .

Лістинг 2.1 – Вміст файлу site-info.def

```
MY_DOMAIN=glite.zntu
###################################
# General configuration variables
###################################
WN_LIST=/opt/glite/yaim/etc/wn-list.conf
# розміщення файлу wn-list.conf
USERS_CONF=/opt/glite/yaim/etc/users.conf
# розміщення файлу users.conf
GROUPS_CONF=/opt/glite/yaim/etc/groups.conf
# розміщення файлу groups.conf
MYSQL_PASSWORD=root
###############################
# Site configuration variables #
```

```
################################
SITE_NAME=zntu_site                    # назва сайту ВО
SITE_EMAIL="vadshkar@yandex.ru"        # e-mail
адміністратора сайту
SITE_LAT=0.0              # дані щодо географічного
SITE_LONG=0.0             # положення сайту
################
APEL_DB_PASSWORD=root     # пароль адміністратора
################
##############################
# CE configuration variables #
##############################
CE_HOST=site.$MY_DOMAIN   # назва хоста з CE
# дані щодо кількості вузлів кластера
CE_CAPABILITY="CPUScalingReferenceSI00=25
Share=zntu_vo:25 Share=zntu_vo:1"
# дані щодо характеристик вузлів
CE_OTHERDESCR="Cores=2,Benchmark=100-HEP-SPEC06"
SE_MOUNT_INFO_LIST="/data/zntu_vo"
############################
# SubCluster configuration
############################
CE_CPU_MODEL=E3200             # модель CPU
CE_CPU_VENDOR=Intel            # виробник CPU
CE_CPU_SPEED=2400              # частота ядра CPU
CE_OS=SL4.8                    # операційна система
CE_OS_RELEASE=2.6.9.           # версія ядра ОС
CE_OS_VERSION="SL4.8"          # версія ОС
CE_OS_ARCH=i686              # архітектура системи
CE_MINPHYSMEM=1024           # обсяг встановленої ОП
CE_MINVIRTMEM=2048           # обсяг віртуальної пам'яті
CE_PHYSCPU=2               # кількість ядер CPU на вузлі
CE_LOGCPU=4                # кількість логічних ядер
CE_SMPSIZE=4               # кількість логічних пар ядер
CE_SI00=1000               # SPEC INT test result
CE_SF00=600                # SPEC FLOAT test result
CE_OUTBOUNDIP=TRUE
CE_INBOUNDIP=TRUE
CE_RUNTIMEENV="GLITE-3_1"
```

```
#    вказувати ці параметри лише на керуючому вузлі
########################################
# Batch server configuration variables
########################################
BATCH_SERVER=$CE_HOST     # хост з менеджером ресурсів
JOB_MANAGER=lcgpbs        # менеджер ресурсів - pbs
CE_BATCH_SYS=torque       # планувальник ресурсів
BATCH_LOG_DIR=/opt/glite/torque_logdir     # стат. дані
BATCH_VERSION=torque-3.1.8-0     # версія планувальника
##############################
# RB configuration variables #
##############################
RB_HOST=$CE_HOST             # хост з брокером ресурсів
##################################
# RGMA configuration variables #
##################################
MON_HOST=$voms.$MY_DOMAIN # хост з системою моніторинга
                          # використання ресурсів
################################
# DPM configuration variables #
################################
DPM_HOST="$CE_HOST"
################################
# BDII configuration variables #
################################
BDII_HOST=$CE_HOST      # хост з інформаційною системою
SITE_BDII_HOST=$CE_HOST
# хости, які покриває інформаційна Система.
# замінити <wn_number> на номер обчислювального вузла
BDII_REGIONS="site wn<wn_number>"
# URL вузлів, задіяних у обчисленні, на які поширюється
дія BDII
BDII_SITE_URL="ldap://site.glite.zntu:2170/zntu_vo=zntu
_vo,o=grid"
BDII_WN<wn_number>_URL="ldap://wn<wn_number>.glite.zntu
:2171/zntu_vo=zntu_vo,o=grid"
SITE_DESC="site.glite.zntu"         # розміщення сайту
SITE_LOC="site.glite.zntu"
SITE_WEB="www.zntu.edu.ua"
SITE_SECURITY_EMAIL="vadshkar@yandex.ru"
SITE_SUPPORT_EMAIL="vadshkar@yandex.ru"
SITE_OTHER_GRID="UGRID"
```

```
VO_SW_DIR=/opt/exp_soft
VOS="ZNTU_VO"
VO_ZNTU_VO_SW_DIR=/opt/exp_soft/zntu_vo
###########
# zntu_vo #
###########
# ВО, порт та сертифікат підключення до неї
VO_ZNTU_VO_VOMS_SERVERS='vomss://voms.glite.zntu:8443/v
oms/zntu?/zntu'
VO_ZNTU_VO_VOMSES="'zntu_vo voms.glite.zntu 15005
/O=zntu/OU=fiot/emailAddress=vadshkar@yandex.ru/L=Zapor
ozhye/ST=baburka/C=UA/CN=Vadim zntu_vo'"
VO_ZNTU_VO_VOMS_CA_DN="'/O=zntu/OU=fiot/emailAddress=va
dshkar@yandex.ru/L=Zaporozhye/ST=baburka/C=UA/CN=Vadim'
"
QUEUES="zntu_vo"
ZNTU_VO_GROUP_ENABLE="zntu_vo"
```

Після виконання зазначених модифікацій файлів, слід перевірити коректність введених даних, виконавши наступну команду:

> source /opt/glite/yaim/etc/gilda/site-info.def

Перевірити, чи визначені всі змінні, необхідні для конфігурування вузлів. Якщо знаходитеся за керуючим вузлом, виконайте наступну команду:

> **/opt/glite/yaim/bin/yaim -v –s /opt/glite/yaim/etc/site-info.def -n BDII_site -n lcg-CE -n TORQUE_server -n TORQUE_utils**

У випадку успіху, виконайте конфігурування:

> **/opt/glite/yaim/bin/yaim -c –s /opt/glite/yaim/etc/site-info.def -n BDII_site -n lcg-CE -n TORQUE_server -n TORQUE_utils**

Якщо знаходитеся за обчислювальним вузлом, виконайте наступну команду:

> **/opt/glite/yaim/bin/yaim -v –s /opt/glite/yaim/etc/site-info.def -n glite-WN -n TORQUE_client**

У випадку успіху, виконайте конфігурування:

> **/opt/glite/yaim/bin/yaim -c –s /opt/glite/yaim/etc/site-info.def -n glite-WN -n TORQUE_client**

У випадку успішного конфігурування вузлів, виведені в терміналі звітні дані будуть мати вигляд як на рис. 2.2.

На керуючому вузлі слід перевірити, чи запущені сервіси локального менеджера ресурсів, ввівши наступну команду:

> **netstat -tnlp**

Позитивна відповідь повинна мати вигляд як наводиться в лістингах 2.2 та 2.3, який вказує на те, що всі необхідні сервіси запущені, а необхідні їм порти – відкриті.

Лістинг 2.2 – Приклад успішного конфігурування вузлів

```
.             /'.-. ')
.     yA,-"-,( ,m,:/ )   .oo.     oo    o      ooo  o.     .oo
.    /      .-Y a  a Y-.     8. .8'    8'8.     8    8b   d'8
.   /          ~ ~ /         8'    .8oo88.     8    8  8'  8
. (_/          '===='         8    .8'     8.   8    8  Y   8
.   Y,-''-,Yy,-.,/           o8o  o8o    o88o  o8o  o8o    o8o
.    I_))_) I_))_)


current working directory: /etc/grid-security/vomsdir/zntu_vo
site-info.def date: Feb 2 17:54 /opt/glite/yaim/etc/site-info.def
yaim command: -c -s /opt/glite/yaim/etc/site-info.def -n BDII_site -n lcg-CE -n TORQUE_server
log file: /opt/glite/yaim/bin/../log/yaimlog
Wed Feb  3 18:19:24 EST 2010 : /opt/glite/yaim/bin/yaim

Installed YAIM versions:
glite-yaim-bdii 4.0.4-6
glite-yaim-clients 4.0.8-7
glite-yaim-core 4.0.8-7
glite-yaim-lcg-ce 4.0.6-1
glite-yaim-torque-client 4.0.3-1
glite-yaim-torque-server 4.0.4  -1
glite-yaim-torque-utils 4.0.4-1
glite-yaim-wms 4.0.7-0

.
.
.
.

Reloading sshd:                                            [  OK  ]
   INFO: Configuration Complete.                                        [  OK  ]
   INFO: YAIM terminated succesfully.
```

Лістинг 2.3 – Запущені сервіси локального менеджера ресурсів

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             Stat
e       PID/Program name
tcp        0      0 0.0.0.0:2119                0.0.0.0:*                   LIST
EN      2717/globus-gatekee
tcp        0      0 0.0.0.0:9002                0.0.0.0:*                   LIST
EN      3282/glite-lb-logd
tcp        0      0 0.0.0.0:40559               0.0.0.0:*                   LIST
EN      3638/maui
tcp        0      0 0.0.0.0:40560               0.0.0.0:*                   LISTEN      3638/maui
tcp        0      0 0.0.0.0:631                 0.0.0.0:*                   LISTEN      2960/cupsd
tcp        0      0 0.0.0.0:15001               0.0.0.0:*                   LISTEN      3567/pbs_server
tcp        0      0 0.0.0.0:2170                0.0.0.0:*                   LISTEN      31158/bdii-fwd
tcp        0      0 0.0.0.0:15002               0.0.0.0:*                   LISTEN      3433/pbs_mom
tcp        0      0 127.0.0.1:2171              0.0.0.0:*                   LISTEN      10427/slapd
tcp        0      0 0.0.0.0:15003               0.0.0.0:*                   LISTEN      3433/pbs_mom
tcp        0      0 127.0.0.1:2172              0.0.0.0:*                   LISTEN      10746/slapd
tcp        0      0 0.0.0.0:15004               0.0.0.0:*                   LISTEN      3638/maui
```

На заключному етапі лабораторної роботи необхідно пересвідчитися, що керуючий вузол зв’язується з відповідним йому обчислювальним вузлом. Для цього слід ввести наступну команду:

> **pbsnodes -a**

Позитивний результат має вигляд, наведений на рис. 2.4.

```
[root@site ~]# pbsnodes -a
wn19
     state = free
     np = 4
     properties = lcgpro
     ntype = cluster
     status = opsys=linux,uname=Linux wn19 2.6.9-89.0.9.EL #
```

Рисунок 2.4 – Результат виконання команди pbsnodes

Наведений лістинг свідчить про те, що керуючий вузол зв’язується з обчислювальним вузлом **wn19**, що даний вузол на момент запиту залишався вільним від обчислень, що він може виконувати 4 обчислювальні потоки; тип організації обчислювальних вузлів – **кластер**. Замість **wn19**, у вашому випадку, має виводитися ім'я хосту, вказане вашим опонентом у файлі **hosts**, у якості імені обчислювального вузла.

2.4 Зміст звіту

2.4.1 Титульний лист.
2.4.2 Мета роботи.
2.4.3 Перелік змінених параметрів в конфігураційних файлах.
2.4.4 Відповіді на контрольні питання.

2.5 Контрольні питання

2.5.1 Що таке **GRID**?
2.5.2 Для чого призначений пакет **gLite**?
2.5.3 Дати визначення **WN** та **CE**.
2.5.4 Як буде виглядати файл **users.conf**, якщо учасники віртуальної організації **vo** об’єднані у групу **group**?
2.5.5 Чи можна у файлі **wn-list.conf**, замість імен хостів (доменних імен) вказувати ір-адреси?

ЛАБОРАТОРНА РОБОТА № 3

Експлуатація створеного кластера

Мета роботи – отримати навички експлуатації обчислювального кластеру під керівництвом системи управління завданнями TORQUE.

3.1 Теоретичні відомості

TORQUE – одна з версій системи PBS (Portable Batch System – система пакетної обробки завдань), що керує завантаженням обчислювальних комплексів. Система пакетної обробки завдань необхідна при одночасному виконанні завдань декількома користувачами на одному обчислювальному комплексі. В результаті використання системи черг та планувальника досягається оптимальне використання обчислювальних ресурсів, а також виключається вірогідність перевантаження вузлів або їх простою. Оскільки в GRID інтенсивність навантаження ресурсів висока, використання TORQUE є виправданим.

Структура TORQUE наведена на рис.3.1.

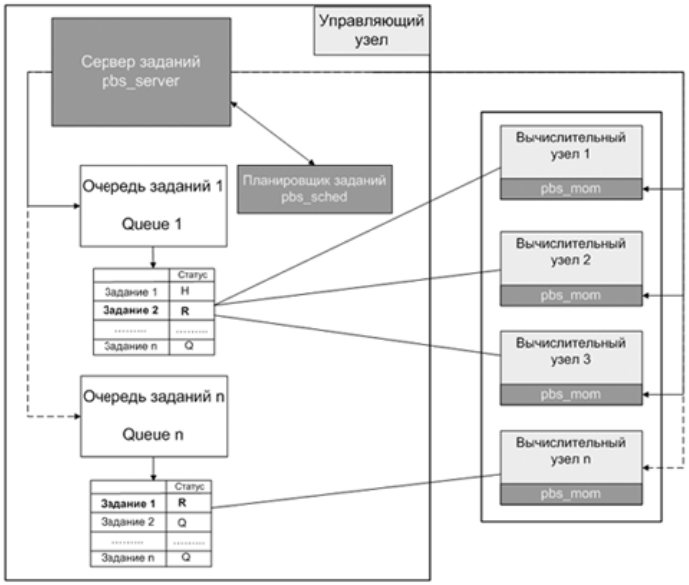


Рисунок 3.1 – Структура TORQUE

Планувальник ставить завдання, що надходять від користувачів, у чергу. Таких черг може бути декілька, вони відрізняються максимальним часом очікування задачі у черзі, максимальним часом виконання, обчислювальними вузлами, що виконують завдання та ін. Під час перебування завдання у черзі йому привласнюється статус. Можливі статуси завдання наведені в табл. 3.1.

Таблиця 3.1 – Можливі статуси завдання у черзі

Статус	Опис
C	Complete – завдання успішно виконане
E	Exit – переривання роботи завдання (вихід з черги)
H	Hold – завдання заблоковане
Q	Queued – завдання поставлене у чергу та очікує виконання
R	Running – завдання виконується
T	Transiting – завдання переміщується у іншу чергу
W	Waiting – завдання очікує виконання особливих умов
S	Suspended – завдання зупинене

TORQUE приймає завдання з керуючого вузла, але авторизовані користувачі можуть підключатися до нього за протоколом SSH з будь-якого іншого комп’ютера у мережі.

3.2 Підготовчий етап

На обчислювальному та керуючому вузлі треба створити групу та обліковий запис, яким дозволено користуватися ресурсами GRID (вони перелічені у файлах groups.conf, users.conf). Для цього необхідно увійти до системи (логін та пароль root) в командному рядку набрати:

```
> groupadd zntu_vo
> useradd zntu_vogrid001 –g zntu_vo
> passwd zntu_vogrid001
> mag
> mag
```

В результаті буде створена група zntu_vo, до неї буде включений користувач zntu_vogrid001 з паролем mag.

Аби CE та WN могли мати можливість обмінюватися файлами (файли завдання, файли з результатами виконання обчислень), між даними вузлами необхідно встановити безпарольний SSH (мережевий протокол сеансового рівня).

На обох вузлах треба увійти до системи під обліковим записом *zntu_vogrid001* та налаштувати безпарольний доступ між вузлами за допомогою протоколу SSH. Нижче наводиться приклад організації безпарольного SSH для вузлів *site.glite.zntu* та *wn19.glite.zntu*.

Приклад організації безпарольного SSH для вузла wn19.glite.zntu:

- заходимо до вузла site.glite.zntu під користувачем zntu_vogrid001:

> ssh zntu_vogrid001@site.glite.zntu

- переходимо в каталог ssh: > cd ~/.ssh
- генеруємо rsa-ключі: > ssh-keygen -t rsa
- на питання задати ім'я файлу, натиснути Enter (ім'я файлу за замовчанням id_rsa), на питання задати пароль, натиснути Enter двічі;
- копіюємо публічний ключ id_rsa.pub на обчислювальний вузол wn19.glite.zntu:

> scp id_rsa.pub zntu_vogrid001@wn19.glite.zntu:~/.ssh

- заходимо до вузла wn19.glite.zntu:

> ssh zntu_vogrid001@wn19.glite.zntu

- переходимо в каталог .ssh: > cd ~/.ssh
- копіюємо публічний ключ id_rsa.pub до списку авторизованих ключів:

> cat id_rsa.pub >> authorized_keys2

- копіюємо закритий ключ з site до wn19 в папку .ssh за допомогою команди scp.

Безпарольний SSH для вузла site.glite.zntu:

- копіюємо зміст папки .ssh з wn19 до site за допомогою команди scp;
- дописуємо authorized_keys2 для site аналогічно вищезазначеному способу.

Організуйте безпарольний доступ між Вашими вузлами, у відповідності з прикладом.

3.3 Порядок виконання роботи

3.3.1 На керуючому вузлі перейти в домашню папку, створити каталог TASK. В цьому каталозі написати програму на Сі, що вирішують наступну задачу: знайти середнє арифметичне мільйона випадкових цілих чисел в діапазоні [0;10000000].

3.3.2 Відкомпілювати програму за допомогою gcc:

> gcc task1.c –o task1

3.3.3 Створити в робочому каталозі файл опису завдання script1.jdl. Текст файлу script1.jdl наведений в лістингу 3.1.

Лістинг 3.1 – Текст файлу - завдання script1.jdl

```
#!/bin/bash
#PBS -l nodes=1,mem=32mb
#PBS -S /bin/bash
#PBS -q zntu_vo
#PBS -o output1.log
#PBS -e errors1.log
scp zntu_vogrid001@site<N>:~/TASK/task1
/home/zntu_vogrid001/task1
sleep 40
./task1
exit 0
```

В цьому завданні запитується один процесор з 32 МБ ОЗП, завдання ставиться у чергу zntu_vo, використовується інтерпретатор bash, потоки stdout та stderr будуть записані на керуючий вузол у файли output.log та errors.log відповідно. Використання команди scp дозволяє скопіювати файл, що виконується, на обчислювальний вузол. Після виконання завдання файли результатів будуть скопійовані на керуючий вузол автоматично, а якщо це, з деяких причин, неможливо – результати будуть збережені на обчислювальному вузлі за адресою /var/spool/pbs/undelivered/.

3.3.4 З керуючого вузла поставити script1.jdl у чергу та заблокувати його виконання:

> qsub script1.jdl -h

В результаті TORQUE привласнить завданню ідентифікатор <task_id> зі статусом «заблоковано».

3.3.5 Переконатися в тому, що завдання заблоковане (має статус H):

> qstat <task_id>

3.3.6 Виконати завдання script2.jdl в інтерактивному режимі. Для цього треба виконати послідовність дій:

- написати файл завдання script2.jdl (його текст наведено в лістингу 3.2).

Лістинг 3.2 – Текст файлу - завдання script2.jdl

```
#!/bin/bash
#PBS -l nodes=1,mem=64mb
#PBS -S /bin/bash
#PBS -q zntu_vo
exit 0
```

- поставити script2.jdl у чергу в інтерактивному режимі:

> qsub script2.jdl –I

- в результаті TORQUE привласнить завданню ідентифікатор <task_id> та підключить консоль користувача до вибраного обчислювального вузла;
- після запиту командного рядка на консолі обчислювального вузла увійти в *mc* та написати програму task2.c, що вирішує наступну задачу: знайти середнє геометричне *N* випадкових цілих чисел в діапазоні [0;10000000], де *N* – вводиться з клавіатури;
- відкомпілювати програму та виконати її, результат роботи програми направити у файл, який потім можна скопіювати на керуючий вузол за допомогою команди *scp*;
- вийти з *mc*, а потім з обчислювального вузла за допомогою команди *exit*.
- поновити виконання завдання, описаного в script1.jdl:

> qrls <task_id>, де <task_id> - ідентифікатор завдання, що був привласнений після виконання команди qsub script1.jdl.

3.3.7 Знову поставити script1.jdl у чергу традиційним способом, переконатися, що завдання виконується та перезавантажити керуючий вузол. Після виконання завдання на обчислювальному вузлі переконатися, що потоки stdout, stderr збережені за адресою /var/spool/pbs/undelivered/.

3.4 Зміст звіту

3.4.1 Титульний лист.
3.4.2 Мета роботи.
3.4.3 Тексти програм та файлів - завдань.
3.4.4 Результати виконання завдань.
3.4.5 Відповіді на контрольні питання.

3.5 Контрольні питання

3.5.1 Переваги виконання завдання в інтерактивному режимі.
3.5.2 Пояснити сенс блокування виконання завдання.
3.5.3 Навіщо потрібний безпарольний доступ між вузлами в контексті обчислювального кластеру?
3.5.4 Яким користувачам дозволено користуватися обчислювальним кластером, як змінити їх перелік?
3.5.5 Що відбувається з результатом виконання завдання у випадку передчасного відключення користувача?

ЛАБОРАТОРНА РОБОТА № 4

Робота з обчислювальним кластером НТУУ "КПІ"

Мета роботи – отримати базові навички роботи з реальним обчислювальним кластером під керівництвом системи управління завданнями PBS.

4.1 Теоретичні відомості

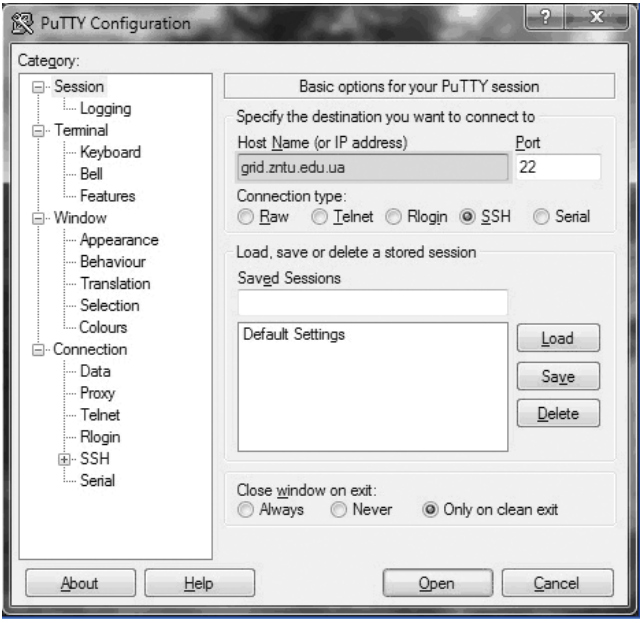
Технічні характеристики кластеру НТУУ КПІ:

- вузли: 44 з двома чотирьох ядерними процесорами Intel Xeon E5440 @ 2.83ГГц та 8 Гб оперативної пам’яті у кожному, а також 68 з двома двох ядерними процесорами Intel Xeon 5160 @3.00ГГц та 4 Гб оперативної пам’яті у кожному;
- мережа обміну даними: InfiniBand;
- дисковий простір: 6 Тб на базі розподіленої файлової системи LustreFS;
- продуктивність: пікова 7 ТФлопс, linpack 5.7 ТФлопс;
- ОС: CentOS release 5.2;
- MPI: openmpi 1.2.8;
- локальний менеджер ресурсів: Torque 2.3.6;
- компілятори C++: intel 10.1, gcc 4.1.2;

- прикладне ПЗ: GROMACS 4.0.2, fftw 3.2, GAMESS;
- протокол доступу: SSH.

4.2 Підготовчий етап

Для забезпечення належного рівню безпеки, робота з віддаленим кластером буде вестись крізь машину-посередник з адресою **grid.zntu.edu.ua**. Для з'єднання з нею необхідно використовувати SSH-клієнт putty. Завантажте операційну систему Windows та запустіть **Putty.** На вкладці Session у полі Host Name (or IP address) ввести адресу **grid.zntu.edu.ua**. Порт **22** — стандартний порт служби SSH. На вкладці Proxy необхідно заповнити поля адреси та порту проксі-сервера ЗНТУ (10.0.2.1 : 8080), та ввести свої дані для авторизації на ньому (поля Username та Password), натиснути кнопку **Open** (рис. 4.1). Після успішного з'єднання на екрані з'явиться привітання з проханням ввести логін і пароль. Авторизація: для авторизації на grid.zntu.edu.ua використовуйте логін **student**, пароль **erF4Xn71**.



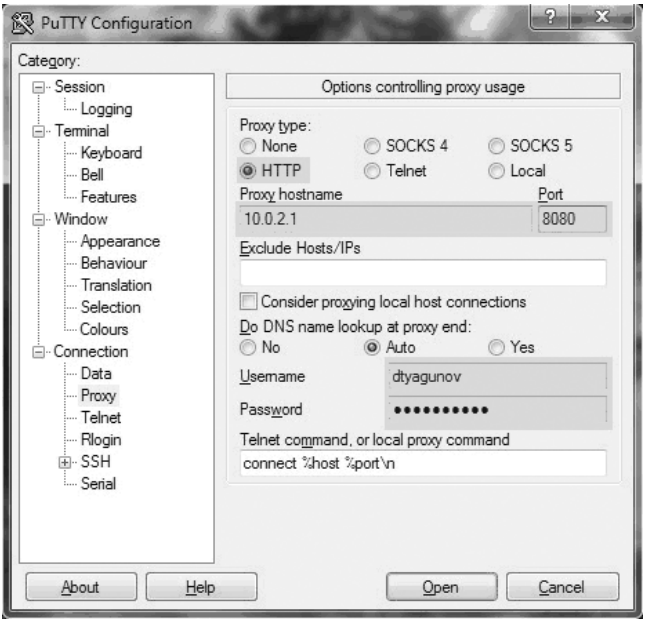


Рисунок 4.1 – Налаштування Putty

При успішній авторизації на екран виведеться привітання (лістинг 4.1).

Лістинг 4.1 – Успішна авторизація

```
login as: student
Using keyboard-interactive authentication.
Password:
Last login: Fri Apr  2 11:31:11 2010 from 188.163.20.177
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserved.


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

  WELCOME TO GRID.ZNTU.EDU.UA

[student@grid] ~ :
```

4.3 Порядок виконання роботи

4.3.1 Перейти до каталогу students (знаходиться в домашній директорії користувача student):

> cd ~/students

4.3.2 Створити свою робочу директорію:

> mkdir <name>

4.3.3 Перейти до робочої директорії:

> cd <name>

4.3.4 Створити файл:

> touch task_<name>.c

4.3.5 Відкрити створений файл у редакторі mcedit (Midnight Commander Editor). Тут <name> — ваше прізвище (маленькі латинські літери):

> mcedit task_<name>.c

4.3.6 Написати програму на мові С, яка підраховує середнє арифметичне суми натуральних чисел в діапазоні від 1 до 1000000, поділене на номер вашого комп'ютера. Отриманий результат вивести на екран. Для компіляції коду програми використовуйте компілятор gcc. За допомогою параметра -o можна вказати вихідний файл для програми:

> gcc task_<name>.c -o task_<name>

4.3.7 Якщо програма успішно зкомпільувалася перевірте правильність її роботи (лістинг 4.2):

> ./task_<name>

Лістинг 4.2 – Перевірка роботи програми

```
[student@grid] ~/students/tyagunov : ./task_tyagunov
999882.75
```

4.3.8 Для зручності роботи в каталозі ~/tools знаходяться два скрипта, призначені для пересилання файлів на кластер і, безпосередньо, для з'єднання з ним по протоколу SSH, їх виконання необхідно ініціювати від користовача root (для цього будемо використовувати команду sudo). За допомоги скрипта kpitransfer завантажте свою робочу директорію на кластер (лістинг 4.3):

> sudo /usr/home/student/tools/kpitransfer.sh <name>

Лістинг 4.3 – Пересилання файлів на кластер

```
Transfering tyagunov to NTUU KPI cluster...

task_tyagunov.c                               100%   49     0.1KB/s   00:00
task_tyagunov                                 100% 6634     6.5KB/s   00:00
```

4.3.9 Дочекайтеся завантаження файлів на кластер та за допомогою скрипта kpiconnect.sh, виконайте з'єднання з кластером НТУУ «КПІ»:

> sudo /usr/home/student/tools/kpiconnect.sh

4.3.10 Якщо з'єднання пройшло успішно на екрані з'явиться привітання (лістинг 4.4). Перейдіть у робочий каталог:

> cd ~/students/<name>

4.3.11 Перекомпілюйте програму:

> gcc task_<name>.c -o task_<name>

Лістинг 4.4 – Успішне з'єднання з кластером НТУУ «КПІ»

```
Connecting to NTUU KPI cluster...

 HHHH   HHHH  PPPPPPPPP    CCCCCCCC    CCCCCCCC
 HHHH   HHHH  PPPP  PPPP  CCCC  CCCC  CCCC  CCCC
 HHHH   HHHH  PPPP   PPPP CCCC   CC   CCCC   CC
 HHHHHHHHHHH  PPPP  PPPP  CCCC        CCCC
 HHHHHHHHHHH  PPPPPPPPP   CCCC        CCCC
 HHHH   HHHH  PPPP        CCCC   CC   CCCC   CC
 HHHH   HHHH  PPPP        CCCC  CCCC  CCCC  CCCC
 HHHH   HHHH  PPPP         CCCCCCCC    CCCCCCCC

 Welcome to cluster at NTUU "KPI"!



 Note from admins:
   Please, run your time comsuming tasks only through
  local resource manager (pbs). All resource greedy
  processes that run bypassing LRMS will be killed as soon
  as detected.

 Check news at http://hpcc.org.ua

 Have a nice day!
```

4.3.12 Перегляньте список доступних черг:

> qstat -q

Зверніть увагу на параметр Walltime. Він вказує максимально можливий час виконання програми у даній черзі. Якщо програма не

встигне відпрацювати у цей період часу її виконання буде перервано і вона буде вилучена з черги.

4.3.13 Вивести загальне число задач у всіх чергах:

> qstat | grep ".pbs" -c

4.3.14 Вивести кількість завдань, що виконуються на даний момент часу:

> qstat | grep " R " -c

4.3.15 Вивести кількість завдань, що очікують виконання:

> qstat | grep " Q " -c

Для того щоб запускати ваші завдання на обчислювальному кластері, необхідно користуватися системою управління завданнями PBS. На практиці це відбувається так: ви готуєте файл завдання PBS і ставите його в (ту чи іншу) чергу PBS використовуючи команду qsub. При цьому ви запитуєте необхідні для цього завдання ресурси: число вузлів кластера, число процесорів на кожному з них, необхідну кількість оперативної пам'яті і час. Якщо запитані ресурси не суперечать наявним налаштуванням кластеру та черги (тобто, ви не просите, наприклад, більше процесорів ніж їх фізично є), завдання буде поставлено в чергу. Після цього, як тільки утворюються вільні ресурси на кластері, відповідні запитуваним, PBS запустить завдання. Після його завершення, файли виводу будуть доставлені в домашній каталог користувача – господаря завдання. Файл завдання являє собою звичайний командний файл (краще всього BASH), який, зазвичай, налаштовує потрібні змінні середовища (колії, розташування тимчасових каталогів, файлів з даними і виведенням) і запускає вашу прикладну програму. Крім того, всередині файлу ви можете отримати дані від PBS через її спеціальні змінні середовища.

4.3.16 Створіть файл - завдання PBS:

> touch script_<name>.job
> mcedit script_<name>.job

4.3.17 Внесіть у нього текст, наведений у лістингу 4.5.

Лістинг 4.5 – Текст файлу - завдання

```
#!/bin/bash
##запитуємо 1 ноду, 1 процесор на ній та 32 Мб пам'яті
#PBS -l nodes=1,mem=32mb
##оболонка bash
#PBS -S /bin/bash
```

```
##черга short
#PBS -q short
##перенапрямлення стандартного потоку виведення
#PBS -o output.log
## перенапрямлення стандартного потоку помилок
#PBS -e errors.log
echo "Running on: $PBS_O_HOST"
echo "Start: `date '+%d.%m.%y %H:%M:%S:%s'`"
echo "Program result: "
##виконання програми
~/students/<name>/task_<name>
echo "End: `date '+%d.%m.%y %H:%M:%S:%s'`"
##затримка 3 хвилини
sleep 180
##вихід з нульовим кодом
exit 0
```

4.3.18 Постановка завдання в чергу:

> qsub script_<name>.job

При успішній постановці завдання в чергу на екран виведеться її унікальний ідентифікатор у форматі <jobid>.pbs.kpi (на це може знадобитися деякий час, тому що планувальнику PBS необхідно дочекатися моменту, коли з'являться вільні запитувані ресурси).

4.3.19 Моніторинг стану завдань у черзі і керування ними:

- детальна інформація про стан виконання:

> qstat -f <jobid>

- загальна інформація про стан виконання:

> qstat <jobid>

Зверніть увагу на поля job_state — поточний статус завдання, exec_host — список нод і номер процесора ноди, на якому виконується завдання.

4.3.20 Після завершення виконання завдання (команда qstat надрукує повідомлення про те, що не знає завдання з ідентифікатором <jobid>: qstat: Unknown Job Id 1150056.pbs.kpi) в вашому робочому каталозі з'являться два файли output.log та errors.log, які містять результуючу інформацію та/або інформацію про виникнення помилок в процесі виконання. За бажанням можете перенести свою робочу директорію з кластеру на grid.zntu.edu.ua. Для цього виконайте команду:

> ~/tools/zntutransfer.sh <name>

4.3.21 Ваша робоча директорія буде скопійована на grid.zntu.edu.ua у папку /usr/home/student/students/<name>/fromkpi.

4.3.22 Для завершення роботи на кластері виконайте команду exit двічі:

> exit

4.4 Зміст звіту

4.4.1 Титульний лист.

4.4.2 Мета роботи.

4.4.3 Тексти програми та файлу - завдання.

4.4.4 Результат виконання команди qstat -f <jobid>.

4.4.5 Відповіді на контрольні питання.

4.5 Контрольні питання

4.5.1 Поясніть необхідність обчислення задач на віддаленому кластері.

4.5.2 Основні можливості програми Putty.

4.5.3 Принципи формування черг PBS.

4.5.4 Команди постановки задачі у чергу, перегляду статусу задачі, зупинки виконання, зняття з черги.

ЛАБОРАТОРНА РОБОТА № 5

Моделювання інфраструктури *Grid* у середовищі *GridSim*

Мета роботи – отримати навички роботи з інструментарієм моделювання *Grid*-систем *GridSim*.

5.1 Теоретичні відомості

Для побудови високоефективної *Grid*-інфраструктури на етапі проектування широко використовуються різноманітні засоби інструментальних комплексів моделювання *Grid*. Одним із таких комплексів є *GridSim* [7, 8]. Використання *GridSim* дозволяє, зокрема, перевіряти кількісні та якісні показники нових алгоритмів роботи брокерів ресурсів, поліпшувати їх тощо.

Безпосереднє використання *Grid*-системи для вирішення оптимізаційних задач (наприклад балансування обчислювального навантаження), що передують проведенню експериментальних досліджень, не завжди є доцільним та обґрунтованим. Це може обумовлюватися наступними факторами:

– мінливістю інфраструктури розподіленої *Grid*-системи, спричиненою самою концепцією *Grid* – варіації складу користувачів ресурсів та їх вимог до ресурсів;

– тимчасовим виходом з ладу окремих компонентів системи тощо.

Перелічені фактори суттєво ускладнюють процес одержання достовірних експериментальних даних. Поліпшення окреслених обставин можливе за рахунок моделювання в середовищі *GridSim*.

Отже, *GridSim* дозволяє здійснювати, зокрема, наступне:

– проводити імітаційне моделювання паралельних та розподілених комп'ютерних систем;

– оцінювати алгоритми роботи брокерів ресурсів (планувальників).

Архітектура *GridSim* базується на трьох наступних ієрархічних рівнях (стратах):

– *JVM* (*Java Virtual Machine*) – нижній ієрархічний рівень;

– підсистема моделювання *SimJava* (набір *Java*-класів) [9];

– набір шаблонів для моделювання компонентів *Grid*-інфраструктури – верхня страта.

В роботі акцентуємо увагу на підсистемі *SimJava*. *SimJava* є пакетом інструментальних засобів дискретно-подійного імітаційного моделювання паралельних та розподілених систем. Цей пакет створено згідно об'єктно-орієнтованого підходу до програмування. У зв'язку із цим, відповідно до концепції поліморфізму, ключовим у *SimJava* є поняття сутності (*entity*), представлене *Java*-класом `Sim_entity` пакету `eduni.simjava`. Певна множина однотипних компонентів досліджуваної системи представляється в *SimJava* окремим класом, що наслідує клас `Sim_entity`.

Загалом процес створення моделі компоненту системи можна представити послідовністю наступних етапів:

1) розширення класу `Sim_entity`;

2) додавання до створюваної моделі сутності потрібної кількості вхідних та вихідних портів, з метою подальшого встановлення міжкомпонентних зв'язків;

3) визначення операційної семантики сутності – механізмів обробки повідомлень.

Отже, для створення моделі окремого компоненту зазначений клас будемо наслідувати новим дочірнім класом компоненту. Для подальшого встановлення міжкомпонентних зв'язків скористаємося класом `Sim_port` цього ж пакету (лістинг 5.1).

Лістинг 5.1 – Приклад створення *SimJava*-моделі *Grid*-компоненту

```
import eduni.simjava.*;
class SomeComponent extends Sim_entity {
     private Sim_port in, out;
     SomeComponent(String name) {
          super(name);
          out = new Sim_port("out");
          in = new Sim_port("in");
          add_port(in);
          add_port(out);
     }
     @Override
     public void body() {...}
}
```

У лістингу 5.1 визначено клас `SomeComponent`, що розширює клас `Sim_entity`. Інтерфейс сутності представлено полями `in` та `out` типу `Sim_port`. Порти моделі компоненту додано в конструкторі класу за допомогою методу `add_port`.

Наслідування класу `Sim_entity` обумовлює потребу перевизначення ключового методу `body()`, тіло якого призначене для опису операційної семантики сутності, на що вказує анотація `@Override` (лістинг 5.2).

Операційна семантика сутності загалом полягає у виконанні наступних дій:

– створенні обробників повідомлень та безпосередньо повідомлень, що мають бути надіслані через вихідний порт (порти) сутності;

– створенні обробників повідомлень, що надходять через вхідний порт (порти).

Лістинг 5.2 – Типовий шаблон методу `body()`

```
@Override
public void body() {
     for(int i=0; i<jobs_count; i++) {
         sim_schedule(out, delay_1, i);
         sim_trace(Sim_system.get_trc_level(),
             "job_"+i+" has been sent...");
         sim_pause(delay_2);
     }
     Sim_event event = new Sim_event();
     sim_wait(event);
     if(event.from_port(in)) {
         sim_trace(Sim_system.get_trc_level(),
             ">> The result has been received!");
         System.out.println("Done!");
         sim_completed(event);
     }
}
```

У лістингу 5.2 обробка повідомлень, що надходять до вхідного порту, здійснюється в циклі `for`. Метод `public void sim_schedule(int dest, double delay_1, int tag)` класу `Sim_entity` призначений для відправлення повідомлень заданого

типу (позначених ідентифікатором `tag`) через порт `dest` із затримкою передачі `delay_1`. На відміну від нього, метод `public void sim_pause(double delay_2)` встановлює інтервали (затримки) між відправленнями повідомлень. Метод `sim_trace` призначений для занесення інформації про відправлення до файлу-лістингу *sim_trace*, що знаходиться в кореневому каталозі проекту. Механізм одержання повідомлень із вхідних портів моделі компоненту наступний:

1) створюється екземпляр класу `Sim_event`, що передається у якості аргументу методу `public void sim_wait(Sim_event event)` класу `Sim_entity`, з метою забезпечення очікування повідомлень на вхідних портах; екземпляр класу `Sim_event` символізує собою подію надходження повідомлення на вхідний порт;

2) за допомогою `if`-конструкції та методу `public boolean from_port(Sim_port p)` класу `Sim_event` організується механізм виявлення події надходження повідомлення до певного порту `p`;

3) сигналізація про завершення обробки повідомлення викликом методу `public void sim_completed(Sim_event event)` класу `Sim_entity`.

Отже, визначений клас `SomeComponent` (лістинг 5.1) дозволить створювати екземпляри *Grid*-компонентів із одним вхідним та одним вихідним портами, а перевизначення в ньому методу `body()` – задавати обробники повідомлень (лістинг 5.2).

Для побудови *SimJava*-моделі *Grid*-інфраструктури на основі визначених сутностей (моделей компонентів) необхідно виконати наступне:

– створити моделі компонентів;

– встановити міжкомпонентні зв’язки – створення архітектурної складової моделі.

Для встановлення міжкомпонентних зв’язків та здійснення дискретно-подійного імітаційного моделювання використовуються методи класу `Sim_system`:

– `public static void initialise()` – ініціалізація моделі системи – засіб переведення моделі із невизначеного стану – у початковий; метод виконує роль конструктора;

– `public static void link_ports(String srcEntity, String outPort, String dstEntity, String inPort)` – засіб встановлення міжкомпонентних зв’язків, де `srcEntity` – символьний

рядок, що ідентифікує модель сутності, з вихідного порту `outPort` якої будуть надходити повідомлення на вхідний порт `inPort` моделі іншої сутності, представленої символьним рядком `destEntity`;

– `public static void set_trace_detail(boolean defTrace, boolean entityTrace, boolean eventTrace)` – засіб форматування контенту файлу-лістингу про здійснення процесу моделювання, де значення `true` прапорця `defTrace` сигналізує про потребу включення до файлу-лістингу *sim_trace* інформації про всі події, які відбулися під час моделювання; значення `true` прапорця `entityTrace` свідчить про потребу включення до лістингу інформації про події, обумовлені лише моделями сутностей; істинне значення прапорця `eventTrace` визначає потребу фіксації лише подій визначеного типу;

– `public static void run()` – засіб запуску процесу імітаційного моделювання.

Підсумовуючи вищесказане, слід зазначити, що дії, пов'язані із налаштуваннями та запуском імітаційного моделювання виконуються у `main`-методі окремого *Java*-класу. Їх можна охарактеризувати наступною послідовністю етапів (лістинг 5.3):

1) ініціалізація моделі;

2) створення екземплярів визначених класів сутностей;

3) встановлення міжпортових зв'язків між створеними екземплярами класів – формування архітектури моделі;

4) регламентування формату лістингу про хід моделювання;

5) запуск моделювання.

Лістинг 5.3 – Шаблон `main`-методу для налаштування та запуску імітаційного *SimJava*-моделювання

```
import eduni.simjava.*;
public class EntryPoint {
     public static void main(String[] args) {
         Sim_system.initialise();
         SomeComponent sc1, sc2;
         sc1 = new SomeComponent("src");
         sc2 = new SomeComponent("dst");
         Sim_system.link_ports("src","out","dst","in");
         Sim_system.set_trace_detail(false,true,false);
         Sim_system.run();}}
```

В лістингу 5.3 комбінацією значень аргументів методу `set_trace_detail` вказано, що результуючий файл *sim_trace* має містити інформацію лише про події, обумовлені екземплярами визначених сутностей.

5.2 Порядок виконання роботи

5.2.1 Побудова *SimJava*-моделі *Grid*-системи.

Потрібно створити *SimJava*-модель *Grid*-системи, що складається з наступних компонентів (рис. 5.1):

– користувацького інтерфейсу *Grid*-ресурсу (компонент *glite-UI* комплексу *gLite*);

– брокеру ресурсів (компонент *glite-CE* комплексу *gLite*);

– обчислювального вузла (компонент *glite-WN* комплексу *gLite*).

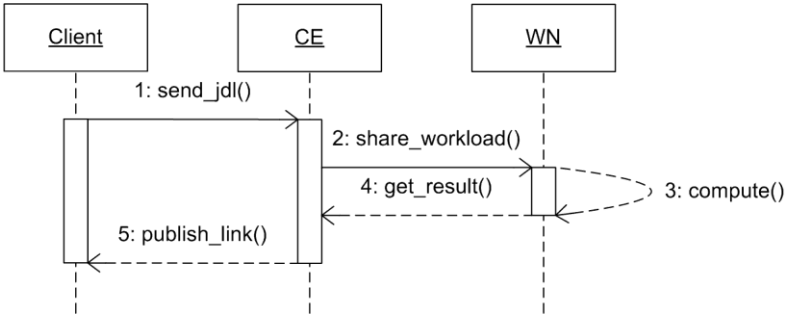


Рисунок 5.1 – *UML*-діаграма послідовності взаємодії компонентів *Grid*

5.2.1.1 В середовищі *IDE NetBeans 8.0* створити новий проект (*create Simple Java Application*).

5.2.1.2 До створеного проекту слід підключити *jar*-архів *simjava.jar*, що надасть змогу використовувати для здійснення моделювання вищерозглянуті класи пакету `eduni.simjava` та відповідні методи.

5.2.1.3 Згідно рис. 5.1, включити до проекту чотири *Java*-файли, з яких три – для класів `Client`, `CE` та `WN` моделей сутностей, а один – для класу-контейнеру `main`-методу – точки запуску моделювання.

Зауваження: класи `Client`, `CE` та `WN` обов'язково мають розширювати клас `Sim_entity` та перевизначати метод `body()`.

5.2.1.4 Створити модель клієнта – користувача *Grid*-ресурсів, що буде виконувати роль генератора заявок на обслуговування – *jdl*-файлів (лістинг 5.4).

Листинг 5.4 – Модель користувача ресурсів

```
package main;
import eduni.simjava.*;
public class Client extends Sim_entity {
    private Sim_port in, out;
    private double delay;
    public static int jobs_count=3;
    Client(String name, double delay) {
        super(name);
        this.delay=delay;
        in = new Sim_port("In");
        out = new Sim_port("Out");
        add_port(in);
        add_port(out);
    }
    @Override
    public void body() {
        for(int i=0; i<jobs_count; i++) {
        sim_schedule(out, 3.0, i);
        sim_trace(Sim_system.get_trc_level(),
            "Job "+i+" from client has been sent...");
        sim_pause(delay);
        }
        Sim_event event = new Sim_event();
        sim_wait(event);
        if(event.from_port(in) && CE.link) {
            CE.link=false;
            sim_trace(Sim_system.get_trc_level(),
             ">> Result link received!");
            System.out.println("Done!");
            sim_completed(event);
}   }   }
```

5.2.1.5 Створити модель компоненту *CE* – брокера ресурсів, що керуватиме роботою обчислювального вузла (лістинг 5.5).

Листинг 5.5 – Модель брокера ресурсів

```
package main;
import eduni.simjava.*;
public class CE extends Sim_entity {
    private Sim_port in1, in2, out1, out2;
    private double delay;
    public static Boolean link = false;
    CE(String name, double delay) {
        super(name); this.delay = delay;
        in1 = new Sim_port("In1");
        in2 = new Sim_port("In2");
        out1 = new Sim_port("Out1");
        out2 = new Sim_port("Out2");
        add_port(in1); add_port(in2);
        add_port(out1); add_port(out2);
    }
    @Override
    public void body() {
    int i=0;
    while(Sim_system.running()) {
        Sim_event event = new Sim_event();
        sim_wait(event);
        if(event.from_port(in1)) {
            sim_process(delay);
            sim_completed(event);
            sim_schedule(out1, 1.0, i);
            sim_trace(Sim_system.get_trc_level(),
        "Workload No."+i+" from CE has been sent...");
            ++i;
        }
        if(event.from_port(in2) && WN.flag) {
            WN.flag=false;
            sim_process(delay/2);
            sim_completed(event);
            link=true;
            sim_trace(Sim_system.get_trc_level(),
        "The output directory generated!");
            sim_schedule(out2, 3.0, ++i);
        } else continue;
     }
     }
}
```

5.2.1.6 Створити модель компоненту *WN* (лістинг 5.6).

Листинг 5.6 – Модель обчислювального вузла

```
package main;
import eduni.simjava.*;
public class WN extends Sim_entity {
    private Sim_port in, out;
    private double delay;
    public int jobs_count=0;
    public static Boolean flag=false;
    WN(String name, double delay) {
        super(name);
        this.delay=delay;
        in = new Sim_port("In");
        out = new Sim_port("Out");
    add_port(in); add_port(out);
    }
    @Override
    public void body() {
       //Boolean flag = false;
       while(Sim_system.running()) {
           Sim_event event = new Sim_event();
           sim_get_next(event);
           if(event.from_port(in)) {
           sim_process(delay);
           sim_completed(event);
           sim_trace(Sim_system.get_trc_level(),
   "Workload No."+jobs_count+" has been processed!");
           if(++jobs_count == Client.jobs_count) {
           //if all jobs have been processed
               flag=true;
           }
       }
       if(flag) {
           sim_schedule(out, 1.0, ++jobs_count);
           sim_trace(Sim_system.get_trc_level(),
             "All the results have been sent to CE!");
       }
       else continue;
       }
    }
}
```

5.2.1.7 Здійснити налаштування моделювання (лістинг 5.7).

Листинг 5.7 – Результуюча модель *Grid*-інфраструктури

```
package main;
import eduni.simjava.*;
public class Grid {
    public static void main(String [] args) {
         Sim_system.initialise();
         Client client = new Client("Client", 10.0);
         CE ce = new CE("CE", 2.0);
         WN wn = new WN("WN", 50.0);
      Sim_system.link_ports("Client","Out","CE","In1");
      Sim_system.link_ports("CE","Out1","WN","In");
      Sim_system.link_ports("WN","Out","CE","In2");
      Sim_system.link_ports("CE","Out2","Client","In");
      Sim_system.set_trace_detail(false, true, false);
         Sim_system.run();
    }
}
```

5.2.2 Проведення імітаційного *SimJava*-моделювання.

Проаналізувати хід моделювання, запустивши створений проект на виконання. Результатом запуску буде файл лістингу *sim_trace* (лістинг 5.8).

Листинг 5.8 – Результати моделювання

```
u: Client at 0.0: Job 0 from client has been sent...
u: CE at 5.0: Workload No.0 from CE has been sent...
u: Client at 10.0: Job 1 from client has been sent...
u: CE at 15.0: Workload No.1 from CE has been sent...
u: Client at 20.0: Job 2 from client has been sent...
u: CE at 25.0: Workload No.2 from CE has been sent...
u: WNs at 56.0: Workload No.0 has been processed!
u: WNs at 106.0: Workload No.1 has been processed!
u: WNs at 156.0: Workload No.2 has been processed!
u: WNs at 156.0: All the results have been sent to CE!
u: CE at 158.0: The output directory generated!
u: Client at 161.0: >> Result link received!
```

5.2.3 Побудувати *SimJava*-модель *Grid*-системи, представленої UML-діаграмою послідовності взаємодії (рис. 5.2).

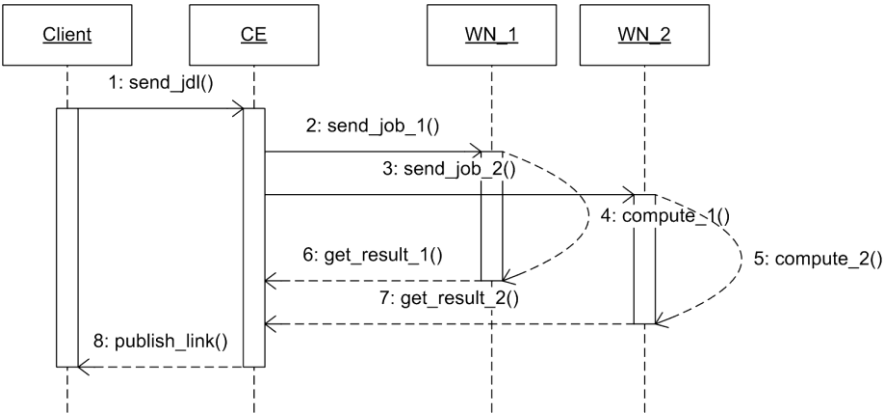


Рисунок 5.2 – Модель *Grid*-системи з двома WN-вузлами

Результат відповідного моделювання має відповідати лістингу 5.9.

Листинг 5.9 – Результати моделювання згідно рис. 5.2

```
u: Client at 0.0: jdl-file has been sent...
u: CE at 10.0: Workload No.0 from CE has been sent...
u: CE at 12.0: Workload No.1 from CE has been sent...
u: WN_1 at 20.0: Workload No.0 has been processed!
u: WN_2 at 22.0: Workload No.1 has been processed!
u: CE at 23.0: The output directory generated!
u: Client at 24.0: >> Result link received!
```

5.3 Зміст звіту

5.3.1 Титульний лист.

5.3.2 Мета роботи.

5.3.3 Фрагменти програмного коду, створеного для виконання завдання 5.2.3.

5.3.4 Відповіді на контрольні питання.

5.4 Контрольні питання

5.4.1 Призначення інструментарію *GridSim*.

5.4.2 Архітектура *GridSim*. Ієрархічні рівні.

5.4.3 Призначення підсистеми моделювання *SimJava*.

5.4.4 Клас *Sim_entity*. Призначення. Методи класу.

5.4.5 Клас *Sim_event*. Призначення. Методи класу.

5.4.6 Клас *Sim_system*. Призначення. Методи класу.

5.4.7 Механізм одержання повідомлень на вхідних портах *SimJava*-моделі компоненту.

5.4.8 Процедура відправлення повідомлень через вихідні порти *SimJava*-моделі компоненту.

5.4.9 Прокоментувати відмінності аргументів-затримок методів `sim_schedule` та `sim_pause` класу `Sim_entity`.

5.4.10 Прокоментувати залежність контенту файлу-лістингу *sim_trace* про хід моделювання від значень аргументів методу `set_trace_detail` класу `Sim_system`.

5.4.11 Послідовність дій для налаштування та запуску імітаційного *SimJava*-моделювання.

ЛАБОРАТОРНА РОБОТА № 6

Створення GRID кластеру на Java та JPPF

Мета роботи – навчитися розгортати та конфігурувати обчислювальний кластер для Java-додатків на базі фреймворка JPPF.

6.1 Теоретичні відомості

JPPF – це багатоплатформова технологія, що дозволяє додаткам, які вимагають великих обчислювальних потужностей, запускатися на будь-якій кількості комп'ютерів з метою зменшення витрат часу на обчислення. Це досягається шляхом розділення додатків на невеликі частини, які можуть виконуватися на різних машинах.

Основні переваги цієї технології:

– просте використання в клієнтських додатках;

– швидкий процес розгортання та налаштування кластера;

– працює на будь-якій системі, що підтримує Java (на Unix/Linux, Windows, Mac OS та інших системах);

JPPF GRID складається з наступних трьох частин (рисунок 6.1):

– клієнти – додатки, що надають завдання на виконання;

– вузли – виконують отримані завдання;

– сервери – компоненти, що отримують завдання від клієнтів, розподіляють їх на нодах, отримують результати виконання завдань та повертають їх клієнтам.

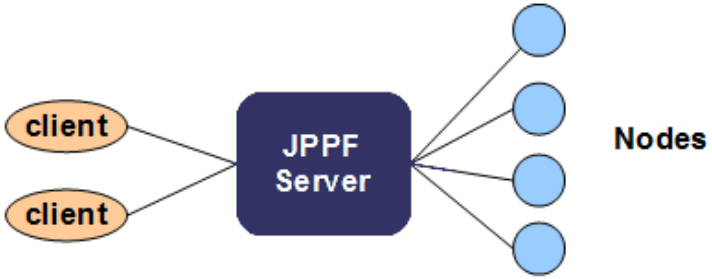


Рисунок 6.1 – Архітектура JPPF GRID кластеру

Базовими термінами в JPPF є робота (job) та завдання (task). При цьому робота складається з одного чи декількох завдань. Кожен вузол може виконувати декілька завдань водночас (за замовчуванням кількість завдань, що виконуються паралельно, дорівнює кількості

логічних процесорів), але лише одну роботу в кожен момент часу. Тому якщо в черзі буде декілька робіт, кожна наступна буде очікувати, поки завершиться попередня робота.

Типовий клієнтський додаток складається з двох частин – класу завдання, який буде виконано на вузлах та клієнту, що буде готувати завдання до виконання. Розглянемо ці частини детальніше.

На лістингу 6.1 наведено приклад коду класа із завданням.

Лістинг 6.1 – Клас CommonTask

```
import org.jppf.node.protocol.AbstractTask;

public class CommonTask extends AbstractTask<Integer> {
  private int a;
  public CommonTask(int a) { this.a = a;}
  @Override public void run() {
    // Обчислення на віддаленому вузлі
    setResult(a);
  }
}
```

Кожний клас із завданням JPPF наслідує базовий клас `AbstractTask<?>`. В дужках вказується клас повертаємого завданням значення. Він має відповідати типу значення, що буде вказано в параметрі до метода `setResult(…)`.

В класі завдання може бути описаний конструктор, за допомогою якого можна буде вказати початкові значення для кожного завдання окремо. Він не є обов'язковим.

Клас повинен перевизначити метод `public void run()`, бо саме він буде виконуватись на обчислювальному вузлі. В цьому методі можна виконувати будь-які дії, після яких має бути повернуто результат за допомогою метода `setResult(result)`.

На лістингу 6.2 наведено приклад коду клієнта:

Лістинг 6.2 – Клас CommonClient

```
import java.util.List;
import org.jppf.client.JPPFClient;
import org.jppf.client.JPPFJob;
import org.jppf.node.protocol.Task;

public class CommonClient {
```

```
  public static void main(String[] args) {
    try (JPPFClient jppfClient = new JPPFClient()) {
      JPPFJob job = new JPPFJob();
      CommonTask task = new CommonTask(5);
      job.setBlocking(true);
      job.add(task);

      List<Task<?>> results =
                    jppfClient.submitJob(job);

      for (Task<?> res: results) {
        if (res.getThrowable() != null)
          System.out.println(res.getId() +
                  " - exception: " +
                  res.getThrowable().getMessage());
        else
          System.out.println(res.getId() +
                  " - result: " + res.getResult());
      }
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

Розглянемо порядок роботи клієнта.

`try (JPPFClient jppfClient = new JPPFClient())` – створення ресурсу jppfClient, який буде доступний в блоці try … catch і буде автоматично звільнений за його межами.

`JPPFJob job = new JPPFJob()` – створення об'єкту роботи.

`CommonTask task = new CommonTask(5)` – створення об'єкту завдання. Метод `job.setBlocking(true)` задає синхронний режим роботи (очікування результатів виконання всіх завдань). Далі завдання за допомогою методу `job.add(…)` додається до черги роботи.

`jppfClient.submitJob(job)` – передача роботи на сервер та очікування результатів.

Результати повертаються від сервера у вигляді списка з типом `List<Task<?>>`. Елементи цього списку – це результати виконання кожного завдання. Далі клієнт в циклі `for (Task<?> res:`

`results)` виводить на екран результати виконання або виключення, якщо завдання завершилось з помилкою.

6.2 Порядок виконання роботи

Примітка: виконання цієї лабораторної роботи потребує три ПК, на яких будуть знаходитись відповідно серверна, виконуюча (вузлова) та клієнтська частини.

6.2.1. Налаштування серверного комп'ютера.

6.2.1.1. Розпакуйте архів *JPPF-5.0.1-driver.zip* у свою робочу папку.

6.2.1.2. Знайдіть серед розпакованих файлів конфігураційний файл config\jppf-driver.properties. Відкрийте його в текстовому редакторі Notepad++ або Eclipse.

6.2.1.3. Знайдіть в файлі наступні параметри:

```
#jppf.discovery.enabled = true
jppf.load.balancing.algorithm = proportional
jppf.load.balancing.profile = proportional_profile
```

Замініть їх та такі:

```
jppf.discovery.enabled = false
jppf.load.balancing.algorithm = nodethreads
jppf.load.balancing.profile = nodethreads_profile
```

Перший параметр вимикає автоматичний пошук інших серверів в локальній мережі. Останні два вказують політику балансування. В цьому випадку завдання будуть розподілятись між вузлами, що простоюють.

6.2.1.4. Збережіть зміни у файлі та запустіть startDriver.bat.

6.2.2. Налаштування обчислювального комп'ютера.

6.2.2.1. Розпакуйте архів *JPPF-5.0.1-node.zip* у свою робочу папку.

6.2.2.2. Знайдіть серед розпакованих файлів конфігураційний файл config\jppf-node.properties. Відкрийте його в текстовому редакторі Notepad++ або Eclipse.

6.2.2.3. Знайдіть в файлі наступні параметри:

```
#jppf.server.host = localhost
#jppf.discovery.enabled = true
```

Замініть їх та такі:

```
jppf.server.host = WS1
```

```
jppf.discovery.enabled = false
```

Перший параметр вказує адресу або назву комп'ютера, на якому розміщено сервер. Можна вказати або IP-адресу, або доменне ім'я ПК. **Замініть його на назву свого серверного ПК!**

Другий параметр вимикає автоматичний пошук інших серверів в локальній мережі.

6.2.2.4. Збережіть зміни у файлі та запустіть startNode.bat.

6.2.3. Запуск консолі моніторингу сервера на клієнтському комп'ютері.

6.2.3.1. Розпакуйте архів *JPPF-5.0.1-admin-ui.zip* у свою робочу папку.

6.2.3.2. Знайдіть серед розпакованих файлів конфігураційний файл config\jppf-gui.properties. Відкрийте його в текстовому редакторі Notepad++ або Eclipse.

6.2.3.3. Знайдіть в файлі наступні параметри:

```
driver1.jppf.server.host = localhost
#jppf.discovery.enabled = true
```

Замініть їх та такі:

```
driver1.jppf.server.host = WS1
jppf.discovery.enabled = false
```

Замініть значення першого параметру на назву свого серверного ПК!

6.2.3.4. Збережіть зміни у файлі та запустіть startConsole.bat. На екрані з'явиться вікно, в якому мають бути показані сервер та підключений до нього вузол.

6.2.4. Створення та налаштування клієнту, що обчислює довжини результатів факторіалів від 50000 до 50020

6.2.4.1. Розпакуйте архів GridAppTemplate.zip у свою робочу папку.

6.2.4.2. Відкрийте середу розробки Eclipse. Якщо при запуску буде запропоновано вказати путь до робочого місця (workspace), вкажіть папку workspace у робочому каталозі.

6.2.4.3. Імпортуйте розпакований проект у робоче місце. Для цього оберіть пункт меню File > Import. У з'явившемся вікні оберіть General > Existing Projects into Workspace та натисніть Next. Натисніть

Browse навпроти поля “Select root directory” та оберіть директорію з розпакованим раніше шаблоном проекту. Натисніть Finish.

6.2.4.4. Створіть клас *org.zntu.grid.FactorialTask*. Для цього оберіть в меню File > New > Class та вкажіть наступні значення в полях та натисніть Finish:

– Package: *org.zntu.grid*

– Name: *FactorialTask*

6.2.4.5. Наберіть вміст файлу згідно з лістингом 6.3.

Лістинг 6.3 – Клас FactorialTask

```
package org.zntu.grid;
import java.math.BigInteger;
import org.jppf.node.protocol.AbstractTask;

public class FactorialTask
extends AbstractTask<Integer> {
  private long n;
  public FactorialTask(long n) {this.n = n;}

  @Override public void run() {
    BigInteger fact = BigInteger.valueOf(1);
    for (long i = 2; i <= n; ++i)
      fact = fact.multiply(BigInteger.valueOf(i));
    int result = fact.toString().length();
    System.out.println(this.getId() +
       " completed with result digit count " + result);
    setResult(result);
  }
}
```

6.2.4.5. Створіть клас org.zntu.grid.FactorialClient та заповніть його згідно з лістингом 6.4.

Лістинг 6.4 – Клас FactorialClient

```
package org.zntu.grid;

import java.util.List;
import org.jppf.client.JPPFClient;
import org.jppf.client.JPPFJob;
import org.jppf.node.protocol.Task;
```

```
public class FactorialClient {
  public static void main(String[] args) {
    try (JPPFClient jppfClient = new JPPFClient()) {
      JPPFJob job = new JPPFJob();
      job.setName("Factorial job");
      job.setBlocking(true);

      for (int i = 50000; i <= 50020; ++i)
        job.add(new FactorialTask(i))
           .setId("Factorial task (" + i + "!)");

      List<Task<?>> results= jppfClient.submitJob(job);
      for (Task<?> task: results) {
        if (task.getThrowable() != null) {
          System.out.println(task.getId() +
                     ", an exception was raised: " +
                     task.getThrowable().getMessage());
        } else {
          System.out.println(task.getId() +
                             ", result digit count: " +
                             task.getResult());
        }
      }
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

6.2.4.6. Знайдіть в проекті конфігураційний файл config\jppf.properties. Відкрийте його в Eclipse.

6.2.4.7. Знайдіть в файлі наступні параметри:

```
driver1.jppf.server.host = localhost
#jppf.discovery.enabled = true
```

Замініть їх та такі:

```
driver1.jppf.server.host = WS1
jppf.discovery.enabled = false
```

Замініть значення першого параметру на назву свого серверного ПК!

6.2.4.8. Створіть конфігурацію запуску на основі шаблону. Для цього зайдіть в меню Run > Run configurations. У вікні зліва розгорніть пункт “Java Application”, натисніть правою кнопкою миші на конфігурації “Run Grid Application” та оберіть пункт Duplicate. Вкажіть для нової конфігурації наступні параметри:

– Name: *Run Factorial Application*

– Main class: *org.zntu.grid.FactorialClient*

6.2.4.9. Натисніть кнопку Apply та закрийте вікно.

6.2.5. Перевірте роботу створеної програми. Натисніть на трикутник біля значку запуску та оберіть пункт **“**Run Factorial Application”**.** Після цього в консолі Eclipse можна спостерігати процес підключення до серверу.

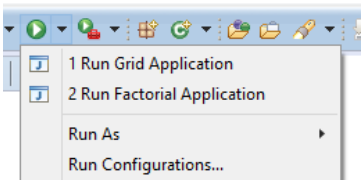


Рисунок 6.2 – Запуск додатка

Тим часом в консолі моніторингу сервера відображається топологія кластеру та які завдання на яких вузлах виконуються.

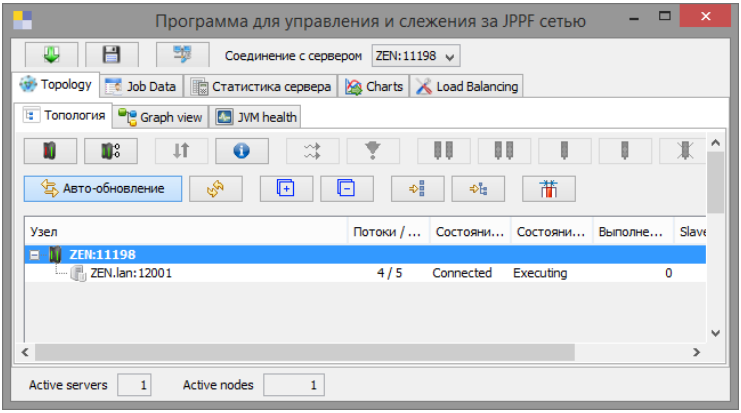


Рисунок 6.3 – Топологія кластеру

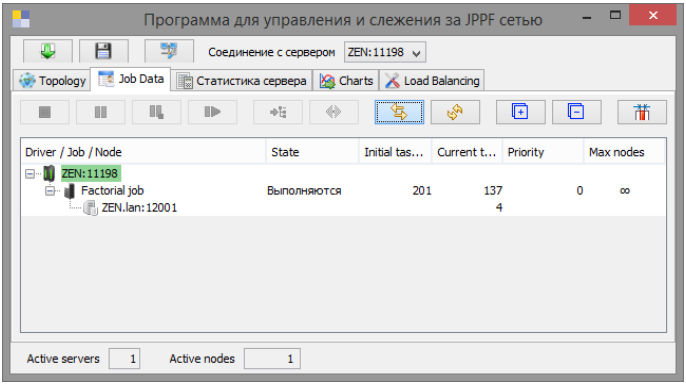


Рисунок 6.4 – Статус робіт та завдань.

6.2.6. Створіть новий клас із завданням, яке підраховує кількість щасливих квитків, які містять 2*N цифр. N передається в конструктор класу. Після цього створіть роботу, що містить завдання розрахунку для N = 1..10 та запустіть їх на кластері.

6.3 Зміст звіту

6.3.1 Титульний лист.

6.3.2 Мета роботи.

6.3.3 Послідовність дій із створення GRID-кластеру.

6.3.4 Фрагменти програмного коду, створеного для виконання завдання 6.2.6.

6.3.5 Відповіді на контрольні питання.

6.4 Контрольні питання

6.4.1 Принцип роботи JPPF.

6.4.2 Як налаштувати мінімальну конфігурацію JPPF-кластера?

6.4.3 Опишіть вміст класів, необхідних для запуску завдання на JPPF-кластері.

ЛАБОРАТОРНА РОБОТА № 7

Розгортання платформи ownCloud

Мета роботи – навчитися розгортати та конфігурувати платформу хмарних обчислень *ownCloud*.

7.1 Теоретичні відомості

Хмарні обчислення (*Cloud Computing*) – сучасна технологія реалізації розподілених обчислень, яка, на відміну від розглянутої раніше *Grid*-технології, призначена здебільшого для її корпоративного застосування.

Визначення хмарних обчислень дається організацією *National Institute of Standard and Technologies* (*NIST*): хмарні обчислення – це модель реалізації зручного суцільного (*ubiquitous*) доступу за вимогою до розподіленого обсягу (*pool*) конфігурованих обчислювальних ресурсів (мережі, сервери, накопичувачі, додатки, сервіси), які можуть бути швидко надані у користування, і так само швидко звільнені, з мінімальними витратами на управління та втручаннями провайдера [11].

Одна з основних переваг від використання технології хмарних обчислень полягає у можливості одержання віддаленого доступу до даних та додатків, розміщених на різноманітних мережних пристроях. Такий доступ можна здійснювати, зокрема, з використанням веб-браузерів.

В роботі виконуватимемо розгортання власної *Cloud*-платформи у *Windows*-середовищі. Для цього скористаємось наступними засобами:

– набір сервісів *Microsoft IIS* (*Internet Information Services*), що включає веб-сервер *IIS* з підтримкою протоколу *CGI* (*Common Gateway Interface*), призначеного для взаємодії веб-сервера із зовнішніми програмними компонентами;

– гіпертекстовий препроцесор *PHP* (*Hypertext Preprocessor*);

– система управління базами даних (СУБД) *MySQL*.

Набір програмних платформ і засобів, для яких було перевірено працездатність *ownCloud* та на основі яких рекомендується виконувати дану роботу, наступний:

– операційна система *Microsoft Windows 7*;

– пакет *ownCloud* – збірка *owncloud-8.0.3.zip* [12];

– препроцесор *PHP* – збірка *php-5.6.9-nts-Win32-VC11-x86.zip*;

– СУБД *MySQL*-сереверу – збірка *mysql-5.5.30-win32.msi*.

Варто відзначити, що у випадку використання операційної системи *Microsoft Windows XP*, версії *PHP 5.5* та наступні вже не підтримуються.

Виконання роботи полягає в інсталяції та налаштування платформи *ownCloud* на прикладі збірки *owncloud-8.0.3.zip*.

Здійсненню підлягають наступні кроки:

– активація сервісів *Microsoft IIS*;

– інсталяція гіпертекстового препроцесора *PHP*;

– інсталяція та налаштування системи *MySQL*;

– розгортання платформи *ownCloud*;

– налаштування платформи *ownCloud*.

7.2 Порядок виконання роботи

7.2.1 Активація сервісів *Microsoft IIS*.

7.2.1.1 Виконати наступні дії: "*Панель управління*" > "*Програми і компоненти*" > "*Увімкнення чи відключення компонентів Windows*". Названа опція при цьому може не з’явитися. Шляхи подолання цієї перепони наступні:

– здійснити "*Увімкнення чи відключення компонентів Windows*" від імені адміністратора: "*Пуск*" > "*Стандартні*" > відкрити командний рядок від імені адміністратора > виконати з командного рядка утиліту *OptionalFeatures.exe*;

– (та/або) відредагувати системний реєстр, виконавши від імені адміністратора у командному рядку команду *regedit* > *HKEY_LOCAL_MACHINE* > *SYSTEM* > *CurrentControlSet* > *Control* > *Windows* > встановити значення змінної *CSDVersion* в 0 та перезавантажити систему.

7.2.1.2 Встановити налаштування сервісів, відзначивши нижченаведені позиції прапорцями (рис. 7.1):

– встановити фільтрацію запитів у якості налаштування безпеки;

– встановити підтримку *CGI* як компоненту розробки додатків;

– задати загальні функції протоколу *HTTP* (*HyperText Transfer Protocol*) – документ за замовчанням, помилки *HTTP*, перегляд каталогу, статичний вміст;

– встановити опції перевірки працездатності та діагностування – ведення журналу *HTTP* та монітор запитів;

– у якості функції підвищення швидкодії встановити "*Стиснення статичного вмісту*";

– у якості засобу керування веб-сайтом задати "*Консоль керування IIS*".

Набуття сили внесених змін займе декілька хвилин.

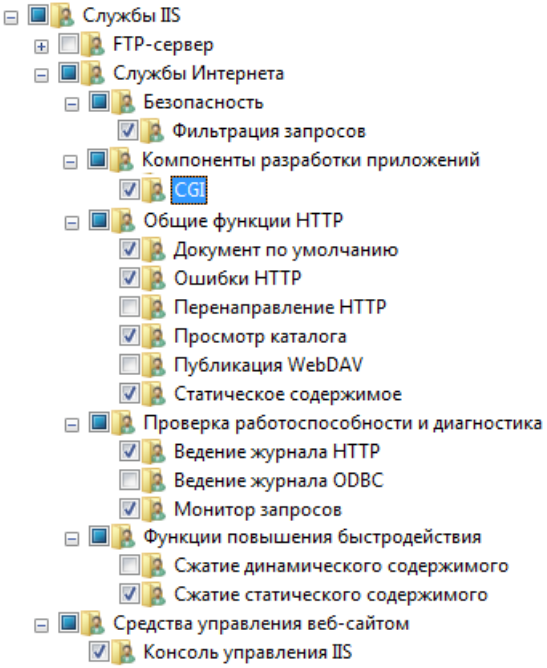


Рисунок 7.1 – Налаштування сервісів *Microsoft IIS*

Підтвердженням успішності активації сервісів *Microsoft IIS* має бути типова відповідна стартова сторінка у вікні веб-браузеру, що доступна за адресою *http://localhost* (рис. 7.2).

Рисунок 7.2 – Стартова сторінка веб-сервера *Microsoft IIS*

7.2.2 Інсталяція і налаштування гіпертекстового препроцесора *PHP*.

7.2.2.1 Розпакувати вміст архіву *php-5.6.9-nts-Win32-VC11-x86.zip* у кореневому каталозі логічного диску *C:*; перейменувати одержаний однойменний каталог у *php*.

7.2.2.2 Модифікувати системну змінну оточення *Path*, додавши до неї шлях *C:\php*:

```
setx path "%path%;c:\php"
```

7.2.2.3 Здійснити перехід до "*Пуск*" > "*Панель керування*" > "*Адміністрування*" > "*Диспетчер служб IIS*".

7.2.2.4 Обрати елемент "*Співставлення обробників*".

7.2.2.5 Заповнити поля форми згідно з рис. 7.3. Одержуваний результат має бути таким, який подано на рис. 7.4.

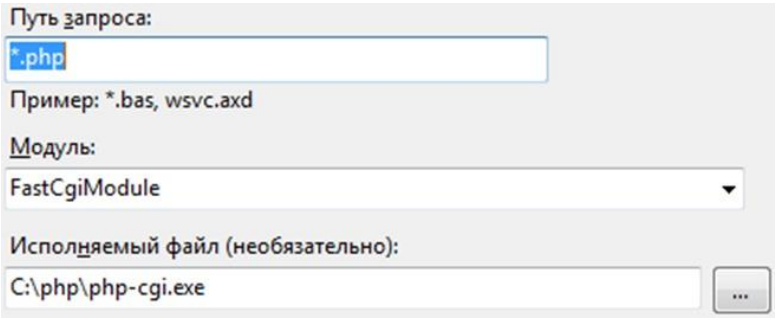


Рисунок 7.3 – Налаштування *PHP* під *CGI*

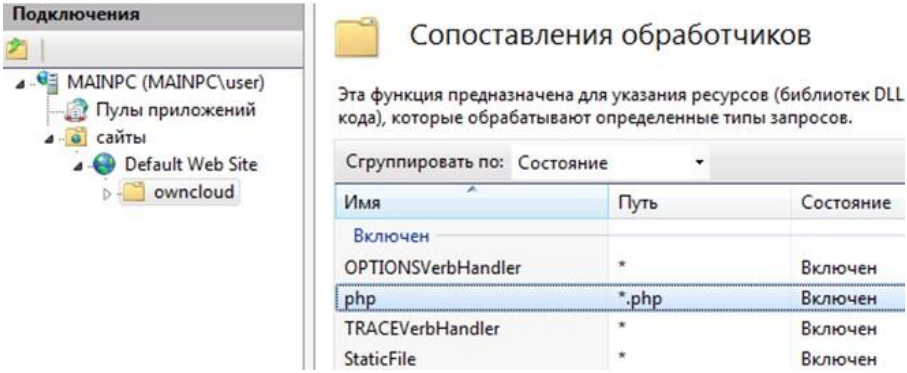


Рисунок 7.4 – Співставлення обробників

7.2.2.6 Встановити сторінку *index.php* у якості стартової (за замовчанням): на панелі інструментів *Диспетчера служб IIS* обрати "*Документ за замовчанням*", для якого задати назву *index.php*.

7.2.2.7 Перевірити зроблені налаштування гіпертекстового препроцесора. Для цього створити файл *test.php* у каталозі *C:\inetpub\wwwroot\owncloud*. Файл повинен мати наступний вміст:

```
<?php phpinfo(); ?>
```

7.2.2.8 У веб-браузері виконати наступний запит:

```
http://localhost/owncloud/test.php
```

Має з'явитися сторінка, наведена на рис. 7.5.

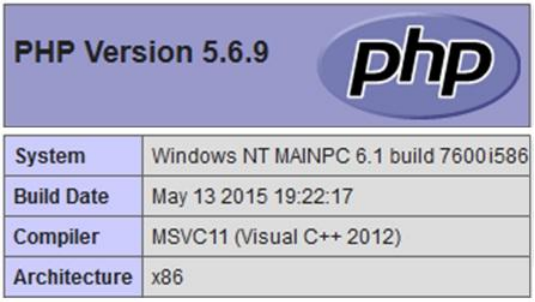


Рисунок 7.5 – Перевірка налаштувань *PHP*

Зауваження: при налаштуванні *PHP* можуть виникнути помилки. Шляхи їх вирішення можуть бути наступними:

– перевірити вміст конфігураційного файлу *web.config*, розміщеного у каталозі *C:\inetpub\wwwroot*;

– внести зміни до конфігураційного файлу *applicationHost.config*, розміщеного у каталозі *C:\Windows\System32\inetsrv\config*:

а) у випадку виникнення помилки `HTTP Error 500.19 - Internal Server Error`, необхідно замінити тег

```
<section name="handlers" overrideModeDefault="Deny" />
```

тегом

```
<section name="handlers" overrideModeDefault="Allow" />
```

б) у випадку виникнення помилки `0x800700b7`, позбутися дублювання тегу `<add accessType="Allow" users="*" />`, та/або закоментувати тег

```
<add name="php" path="*.php" verb="*" modules=
"FastCgiModule" scriptProcessor="C:\php\php-cgi.exe"
resourceType ="Unspecified" />
```

, розміщений між тегами `<handlers accessPolicy="Read, Script">` та `</handlers>`, якщо у файлі *applicationHost.config* вже існує конструкція, наведена в лістингу 7.1.

Лістинг 7.1 – Вміст файлу *applicationHost.config*

```
<location path="Default Web Site/owncloud">
        <system.webServer>
            <handlers>
                <add name="php" path="*.php" verb="*"
modules="FastCgiModule" scriptProcessor="C:\php\php-
cgi.exe" resourceType="Unspecified" />
            </handlers>
        </system.webServer>
</location>
```

– модифікувати файл *php.ini* – скопіювати файл *php.ini-development* з каталогу *C:\php* до каталогу *C:\Windows*, перейменувавши цей файл у *php.ini*; встановити значення змінних перейменованого файлу згідно лістингу 7.2.

Лістинг 7.2 – Вміст файлу *php.ini*

```
log_errors=On
cgi.force_redirect = 0
cgi.fix_pathinfo = 1
fastcgi.impersonate = 1
fastcgi.logging = 0
```

При цьому змінна `open_basedir` має бути закоментована.

7.2.3 Інсталяція і налаштування *MySQL*.

7.2.3.1 Встановити *MySQL*-сервер – збірку *mysql-5.5.30-win32.msi*.

7.2.3.2 Модифікувати вміст файлу *php.ini*, розміщеного у каталозі *C:\Windows*:

– задати значення змінній `extension_dir`: `extension_dir = "C:\php\ext"`;

– розкоментувати рядки, наведені в лістингу 7.3.

Лістинг 7.3 – Модифікація вмісту файлу *php.ini*

```
extension=php_curl.dll
extension=php_gd2.dll
extension=php_mysql.dll
extension=php_mysqli.dll
extension=php_pdo_mysql.dll
```

7.2.4 Розгортання платформи *ownCloud*.

7.2.4.1 Розпакувати вміст архіву *owncloud-8.0.3.zip* до однойменного каталогу.

7.2.4.2 Перейменувати одержаний каталог в *owncloud*.

7.2.4.3 Помістити каталог *owncloud* до каталогу *C:\inetpub\wwwroot*.

7.2.4.4 Змінити права доступу до каталогу *owncloud*: "*Властивості*" > "*Безпека*" > "*Користувачі*" > "*Змінити*" > дозволити запис для групи "*Користувачі*".

7.2.4.5 Перевірити успішність інсталяції *ownCloud*, виконавши у адресному рядку браузера наступний запит:

```
http://localhost/owncloud/
```

У випадку успіху має з'явитися сторінка, подібна до такої, наведеної на рис. 7.6.

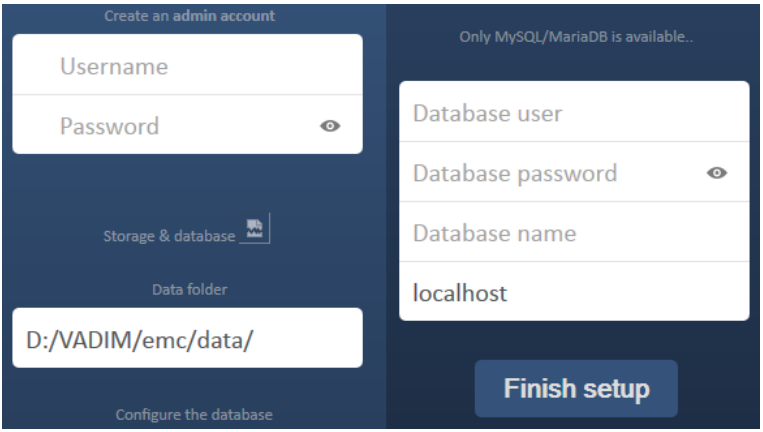


Рисунок 7.6 – Підтвердження успішності інсталяції *ownCloud*

Зауваження: у випадку виникнення помилок можливі наступні шляхи їх вирішення:

– створити пустий каталог *data* у каталозі *C:\inetpub\wwwroot\owncloud*;

– аби позбутися попереджень (*warning*-повідомлень) про непрацездатність *.htaccess*, розміщеного у каталозі *C:\inetpub\wwwroot\owncloud\config*, необхідно модифікувати вміст файлу *config.php*, розміщеного в цьому ж каталозі, додавши елемент `'datadirectory'`, що міститиме шлях до каталогу даних *data*, подібно до того як наведено у лістингу 7.4;

Лістинг 7.4 – Доповнений вміст файлу *config.php*

```
<?php
$CONFIG = array (
  'instanceid' => 'oc8c64jmy7ci',
  'datadirectory' => 'D:/VADIM/emc/data/',
);
```

– створити відповідний каталог *data* за вказаним у файлі шляхом;

– перезапустити веб-сервер *IIS*, виконавши в командному рядку наступну команду:

```
iisreset /restart
```

Знову спробувати виконати завдання 7.2.4.5.

7.2.5 Налаштування платформи *ownCloud*.

7.2.5.1 Заповнити поля форми, наведеної на рис. 7.6.

Зауваження: логін та пароль облікового запису адміністратора *ownCloud* не мають співпадати із логном та паролем СУБД *MySQL*. У першому випадку введемо пару значень (*mag*, *mag*), у другому – (*root*, *root*).

7.2.5.2 У поле "*Database name*" (рис. 7.6) введемо значення *owncloud_db*.

7.2.5.3 Зафіксувати введені дані, натиснувши на елемент управління "*Finish setup*", у результаті чого має з'явитися сторінка-привітання, наведена на рис. 7.7.

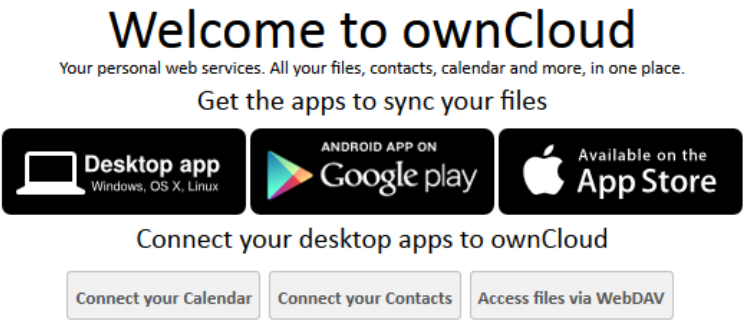


Рисунок 7.7 – Початкова сторінка середовища *ownCloud*

7.2.5.4 Одержати доступ до файлів, розміщених у розгорнутому хмарному середовищі, ввівши в адресному рядку браузера шлях

```
http://localhost/owncloud/index.php/apps/files/
```

, де знаходиться, зокрема, файл з інформацією для користувача – *ownCloudUserManual.pdf*, який переглянути безпосередньо у браузері.

7.3 Зміст звіту

7.3.1 Титульний лист.

7.3.2 Мета роботи.

7.3.3 Знімок екрану, що підтверджує успішність розгортання платформи *ownCloud.*

7.3.4 Відповіді на контрольні питання.

7.4 Контрольні питання

7.4.1 Призначення технології хмарних обчислень.

7.4.2 Прокоментувати відмінність технології хмарних обчислень від технології *Grid*-обчислень.

7.4.3 Компоненти (та їх призначення), необхідні для розгортання системи *ownCloud.*

7.4.4 Етапи розгортання та налаштування системи *ownCloud*.

ЛІТЕРАТУРА

1. Кирьянов А. Введение в технологию Грид / А. Кирьянов, Ю. Рябов. – Гатчина, 2006. – 39с.

2. Шпаковский Г. Применение технологии MPI в Грид / Г.И. Шпаковский, В.И. Стецюренко, А.Е. Верхотуров, Н.В. Серикова. – Минск, 2008. – 137с.

3. GLite documentation / CERN EGEE support [Електронний ресурс]. – Режим доступу: http://glite.web.cern.ch/glite/documentation /default.asp – 2010.

4. Middleware gLite / НТУУ «КПИ» [Електронний ресурс]. – Режим доступа: http://grid.kpi.ua/index.php?option=com_content&task= view&id=28&Itemid=57 – 2010.

5. Система пакетной обработки заданий torque. Руководство пользователя. Т-Платформы, 2008.

6. GRID UA / Центр поддержки пользователей ГРИД НАНУ [Електронный ресурс] – Режим доступа: http://grid.bitp.kiev.ua /index.php?lang=ru – 2010.

7. GridSim: A Grid Simulation Toolkit For Resource Modeling And Application Scheduling For Parallel And Distributed Computing [Електронний ресурс] // University of Melbourne. – Режим доступу: http://www.cloudbus.org/gridsim/.

8. Дорошенко А.Ю. Паралельна розподілена реалізація моделювання паралельних обчислень / А.Ю. Дорошенко, М.В. Гнинюк // Проблеми програмування. – 2014. – № 1. – С. 40 – 48.

9. SimJava [Електронний ресурс] // Institute for Computing Systems Architecture (ICSA), School of Informatics, University of Edinburgh. – Режим доступу: http://www.icsa.inf.ed.ac.uk/research/ groups/hase/simjava/.

10. JPPF documentation [Електронний ресурс] – Режим доступу: http://jppf.org/doc/v5/.

11. Botta A. Integration of Cloud computing and Internet of Things: A survey / A. Botta, W. de Donato, V. Persico, A. Pescape // Future Generation Computer Systems. – 2016. – Vol. 56, Issue C. – P. 684–700. doi: 10.1016/j.future.2015.09.021

12. ownCloud Documentation Overview [Електронний ресурс] – Режим доступу: https://doc.owncloud.org.