Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

Кафедра комп’ютерних

систем та мереж

Звіт

з лабораторної роботи № 2

**“Моделювання інфраструктури Grid у середовищі GridSim”**

Виконав ст. гр. КНТ-514м Школовий В.В.

Прийняв Скрупський С.Ю.

2024р.

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

Хід роботи

Побудувати SimJava-модель Grid-системи, представленої UML-діаграмою послідовності взаємодії.

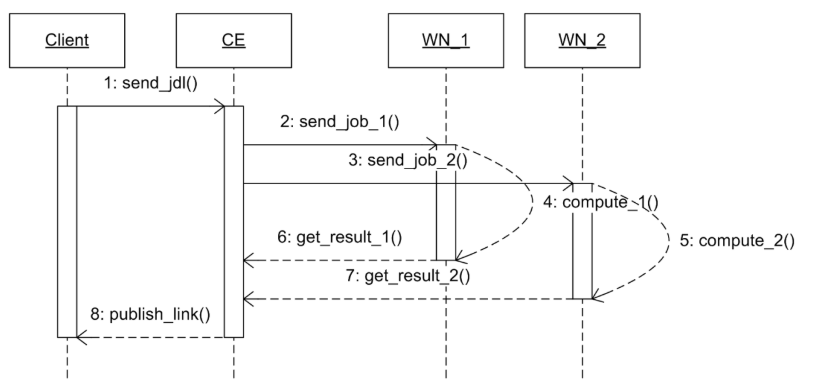


Рисунок 1.1 – Модель Grid-системи з двома WN-вузлами

Було створено чотири java-класи Client, CE, WN та Grid.

Листинг 1.1 – java-клас Client

package main;

import eduni.simjava.\*;

public class Client extends Sim\_entity {

private Sim\_port in, out;

private double delay;

public static int jobs\_count = 2;

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() {

sim\_trace(Sim\_system.get\_trc\_level(), "jdl-file has been sent...");

sim\_schedule(out, 3.0, 0); // Відправка JDL-файлу

for (int i = 0; i < jobs\_count; i++) {

sim\_schedule(out, 3.0, i + 1);

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)) {

sim\_trace(Sim\_system.get\_trc\_level(), ">> Result link received!");

System.out.println("Done!");

sim\_completed(event);

}

}

}

Лістинг 1.2 – java-клас CE

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)) {

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);

}}}}

Лістинг 1.3 – java-клас WN

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() {

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) {

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!");

}

}

}}

Лістинг 1.4 – java-клас Grid

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();

}

}

2. Проведення імітаційного SimJava-моделювання.

Запустивши створений проєкт на виконання. Результатом запуску буде файл лістингу sim\_trace.

Лістинг 2.1 – Результати моделювання

u: Client at 0.0: jdl-file has been sent...

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: WN at 56.0: Workload No.0 has been processed!

u: WN at 106.0: Workload No.1 has been processed!

u: WN at 106.0: All the results have been sent to CE!

u: CE at 108.0: The output directory generated!

u: Client at 111.0: >> Result link received!

u: WN at 111.0: All the results have been sent to CE!

Висновок: під час виконання лабораторної роботи було отримано навички роботи з інструментарієм моделювання Grid-систем GridSim.

Контрольні питання

1. Призначення інструментарію GridSim.

Використання GridSim дозволяє, зокрема, перевіряти кількісні та якісні показники нових алгоритмів роботи брокерів ресурсів, поліпшувати їх тощо.

2. Архітектура GridSim. Ієрархічні рівні.

Архітектура GridSim побудована за ієрархічним принципом, де кожен рівень відповідає за певні функції. Основні ієрархічні рівні GridSim:

1. **Рівень користувача (User Layer)**
2. **Рівень планування (Scheduling Layer)**
3. **Рівень ресурсів (Resource Layer)**
4. **Рівень мережі (Network Layer)**
5. **Базовий рівень (Core Layer)**

3. Призначення підсистеми моделювання SimJava.

SimJava є пакетом інструментальних засобів дискретно-подійного імітаційного моделювання паралельних та розподілених систем. Цей пакет створено згідно об’єктно-орієнтованого підходу до програмування.

4. Клас Sim\_entity. Призначення. Методи класу.

Призначення:

* Sim\_entity представляє базовий клас для моделювання об'єктів, які беруть участь у системі, наприклад, ресурси, вузли, або користувачі.
* Використовується для визначення поведінки сутностей, які обробляють події в симуляції.

Методи класу:

1. Конструктор
2. Методи для управління станом
3. Методи для обробки подій
4. Методи для надсилання/отримання повідомлень
5. Методи доступу

5. Клас Sim\_event. Призначення. Методи класу.

Призначення:

* Використовується для представлення подій, які передаються між сутностями в симуляції.
* Події несуть інформацію про відправника, отримувача, час і додаткові дані.

Методи класу:

1. Конструктор
2. Методи доступу до властивостей події
3. Методи для модифікації

6. Клас Sim\_system. Призначення. Методи класу.

Призначення:

* Основний клас для управління симуляцією.
* Відповідає за запуск, синхронізацію, управління подіями та часом у симуляції.

Методи класу:

1. Ініціалізація системи
2. Робота з сутностями
3. Управління симуляцією
4. Обробка подій
5. Час у симуляції

7. Механізм одержання повідомлень на вхідних портах SimJava-моделі компоненту.

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.

8. Процедура відправлення повідомлень через вихідні порти SimJava-моделі компоненту.

* 1. Ініціалізація вихідного порту: у класі-сутності створіть вихідний порт;
  2. Підключення портів: у main з’єднайте вихідний порт відправника з вхідним портом отримувача;
  3. Відправлення повідомлення: використовуйте метод send() для передачі події через порт;
  4. Ініціалізація вхідного порту: у класі-отримувачі створіть вхідний порт;

### **Обробка отриманих подій:** у методі body() отримуйте та обробляйте події;

* 1. Запуск симуляції: додайте сутності в симуляційну систему, з'єднайте порти, і запустіть;

9. Прокоментувати відмінності аргументів-затримок методів sim\_schedule та sim\_pause класу Sim\_entity.

* sim\_schedule(int dest, double delay, int tag) Затримка delay вказує, через скільки симуляційного часу подія буде надіслана іншій сутності. Використовується для управління подіями між сутностями.
* sim\_pause(double delay) Затримка delay визначає час (у симуляційних одиницях), на який сутність припиняє свою активність. Використовується для імітації простою або очікування.

10. Прокоментувати залежність контенту файлу-лістингу sim\_trace про хід моделювання від значень аргументів методу set\_trace\_detail класу Sim\_system.

1. Метод set\_trace\_detail(int level) визначає рівень деталізації журналу симуляції:

* **level = 0**: мінімальна деталізація (ключові події).
* **level = 1**: середня деталізація (події між сутностями).
* **level = 2**: максимальна деталізація (всі дії, включаючи обробку подій).

1. Від рівня залежить кількість і глибина записів у файлі sim\_trace.

11. Послідовність дій для налаштування та запуску імітаційного SimJava-моделювання.

* Ініціалізувати систему:

Sim\_system.initialise();

* Створити сутності: Ініціалізуйте класи, які розширюють Sim\_entity.
* Додати порти: Використовуйте add\_port() для створення вхідних і вихідних портів.
* З'єднати порти: Налаштуйте зв'язки між сутностями через Sim\_system.link\_ports().
* Додати сутності в систему: Використовуйте Sim\_system.add() (необов’язково, залежить від реалізації).
* Налаштувати журнал (опціонально): Встановіть рівень деталізації через set\_trace\_detail().
* Запустити симуляцію:

Sim\_system.run().