МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРОННИХ ТА ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ

Засоби інтеграції розподілених систем

Методичні вказівки
до виконання лабораторних робіт та самостійної роботи
для здобувачів вищої освіти
спеціальності 121 – "**Інженерія програмного забезпечення**"
рівень вищої освіти – *перший (бакалаврський)*

Обговорено і рекомендовано
на засіданні кафедри інформаційних технологій та програмної інженерії
протокол № 1 від 31.08.2021

Чернігів, НУ «Чернігівська політехніка», 2021

Засоби інтеграції розподілених систем. Методичні вказівки до виконання лабораторних робіт та самостійної роботи для здобувачів вищої освіти спеціальності 121 – "Інженерія програмного забезпечення", рівень вищої освіти – перший (бакалаврський) / Укл.: Білоус І.В. – РВВ НУ «Чернігівська політехніка», 2021. – 40с.

Укладач: Білоус Ірина Володимирівна, кандидат технічних наук, доцент, завідувач кафедри інформаційних технологій та програмної інженерії

Відповідальний за випуск: Войцеховська М.М., доктор філософії, викладач кафедри інформаційних технологій та програмної інженерії

Рецензент: Бичко В.А., кандидат фізико-математичних наук, доцент, доцент кафедри інформаційних та комп’ютерних систем Національного університету "Чернігівська політехніка"

ЗМІСТ

ВСТУП

Лабораторні роботи є сполучною ланкою між лекційними заняттями і самостійною роботою здобувачів ВО. В процесі виконання лабораторних робіт експериментально перевіряються ключові питання курсу «Засоби інтеграції розподілених систем», набуваються практичні навички проектування і реалізації розподіленої взаємодії з використанням засобів інтеграції розподілених систем, перевіряється рівень засвоєння основних положень дисципліни.

Методичні рекомендації до виконання лабораторних робіт орієнтовані на використання засобів інтеграції розподілених систем Java RMI, MPI, Java WSO2 Microservices Framework. Передбачається, що здобувачі знайомі з основами роботи з операційними системами, мовами програмування Java/C++ та володіють англійською мовою технічного рівня. Лабораторні роботи можуть бути виконані за допомогою інших технологій розподіленого програмування за погодженням з викладачем.

Здобувач зобов'язаний до лабораторного заняття прочитати методичні рекомендації до лабораторної роботи і спробувати виконати її самостійно. Під час лабораторного заняття здобувач показує викладачеві результати роботи, проводить консультації з виниклих питань та завершує роботу. Обсяг виконаної роботи може бути різним, залежно від того, на яку оцінку претендує здобувач. Коли робота закінчена, здобувач повинен її захистити. Захист полягає в виконанні практичного завдання, відповіді на питання по темі лабораторної роботи і внесення деяких змін в розроблювану систему в присутності викладача.

По кожній роботі здобувач повинен оформити звіт. Звіти оформляються за допомогою текстового редактора Word на папері формату А4 відповідно до вимог стандартів на оформлення технічної документації. Звіт по роботі є розділом підсумкового документа.

За лабораторну роботу здобувач може отримати до 100 балів, з урахуванням своєчасності і якості виконання всіх складових роботи. Складовими є: звіт, проект, практичне завдання і відповіді на контрольні питання. Оцінки, отримані за лабораторні роботи з відповідними коефіцієнтами враховуються при виставленні семестрової оцінки поточного контролю. Для отримання допуску на іспит всі лабораторні роботи повинні бути виконані і кожна з них оцінена не менше ніж на 60 балів.

Лабораторна робота №1
ОЗНАЙОМЛЕННЯ З ТЕХНОЛОГІЄЮ JAVA RMI

Ознайомлення з основними принципами функціонування технології Java RMI. Реалізація віддалених об'єктів і виклик методів віддалених об’єктів.

1.1 Короткі теоретичні відомості

Java Remote Method Invocation – інтерфейс виклику віддалених методів. Розподілена об'єктна модель, що специфікує, яким чином проводитися виклик віддалених методів, працюючих на іншій віртуальній машині Java.

Додаток RMI часто розглядаються як дві роздільні програми – сервер і клієнт. Типовий серверний додаток створює деякі віддалені об’єкти, робить посилання до них доступними, і чекає від клієнтів виклику методів для цих об’єктів. Типовий клієнтський додаток отримує віддалене посилання до одного або декількох віддалених об’єктів на сервері, а потім викликає для них методи. Система RMI передбачає механізм, за допомогою якого сервер і клієнт зв'язуються один з одним і обмінюються інформацією.

Розподілений додаток, побудований за допомогою Java RMI, створений з інтерфейсів і класів. Інтерфейси визначають методи, а класи реалізують методи, визначені в інтерфейсах, а також, можливо, визначають і нові додаткові методи. У розподіленому додатку передбачається розміщення деяких реалізацій на різних віртуальних машинах. Об’єкти, що мають методи, які можуть викликатися між різними віртуальними машинами, є віддаленими об’єктами – remote objects.

Об'єкт стає віддаленим об’єктом при реалізації віддаленого інтерфейсу remote interface, який має наступні характеристики.

1) Віддалений інтерфейс розширює інтерфейс java.rmi.Remote.
2) Кожен метод інтерфейсу оголошує java.rmi.RemoteException у своїй умові throws, окрім будь-яких інших виключень, специфічних для конкретного додатку.

При передачі об’єкту від однієї віртуальної машини до іншої, RMI розглядає віддалений об’єкт remote object не так, як звичайний, не віддалений об’єкт. Замість того, щоб робити копію реалізації об’єкту в приймаючій віртуальній машині, RMI передає віддалений stub заглушку для віддаленого об’єкту. Цей stub діє як локальне представлення, або заступник (proxy) для віддаленого об’єкту, а для того, що викликає, є віддаленим посиланням. Той, що викликає, викликає метод для локального stub, який відповідає за обробку виклику методу віддаленого об’єкту.

Переданий stub для віддаленого об’єкту реалізує той же набір віддалених інтерфейсів, який реалізує віддалений об’єкт. Це дозволяє для stub бути "передавачем" до будь-якого з інтерфейсів, який реалізує віддалений об’єкт. Проте це також означає, що тільки ті методи, які визначені у віддаленому інтерфейсі, можуть бути викликані на приймаючій віртуальній машині.

1.2 Хід роботи

Розглянемо створення простого застосування на прикладі Hello World. Сервер надає один метод, що приймає як параметр ім 'я і що повертає рядок «Hello ім 'я».

1) Створюємо інтерфейс IHelloWorld.

```
package lab1;
import java.rmi.Remote;
import java.rmi.RemoteException;

// інтерфейс повинний розширювати Remote і генерувати
RemoteException
public interface IHelloWorld extends Remote {
   public String WorHello(String name) throws RemoteException;
}
```

2) Створюємо його реалізацію, клас HelloImpl.

```
package lab1;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements
IHelloWorld {
   /* UnicastRemoteObject надає таку реалізацію багатьох
    * методів класу java.lang.Object (equals, hashCode, toString), відпові-
дні для віддалених об'єктів
   */
   public HelloImpl() throws RemoteException{
      super();
   }
   public String WorHello(String name) throws RemoteException{
      return "Hello " + name;
   }
}
```

3) Створюємо сервер, клас HelloServer.

```
package lab1;
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
```

```
public class HelloServer {
   public static void main(String[] args) {
      // задаємо файл політики безпеки
      System.setProperty("java.security.policy", "D:\\rmi.policy");
      // Вказуємо сервер на якому реєструється ім'я сервісу
      String name = "rmi://localhost/HelloService";
      if(System.getSecurityManager() == null) {
         System.setSecurityManager(new RMISecurityManager());
      }
      try {
         /* створюємо RMI Registry. Якщо його не створювати, то потрібно запускати окремо.
          * RMI Registry є сервісом іменувань і дозволяє клієнтові
          * отримати посилання на віддалений об'єкт по імені
          */
         LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
         IHelloWorld hw = new HelloImpl();
         // реєструємо об'єкт в RMI Registry
         Naming.rebind(name, hw);
         System.out.println("HelloServer bound");
      } catch(Exception e) {
         System.err.println("HelloServer Trouble : " + e.getMessage());
         e.printStackTrace();
      }
   }
}
```

4) Створюємо файл rmi.policy. Шлях до файлу повинний співпадати з вказаним в класі сервера. Вміст файлу:

```
grant {
     permission java.net.SocketPermission "*:1024-65535", "connect, accept";
};
```

5) Створюємо клієнтський клас HelloClient

```
package lab1;
import java.rmi.*;

public class HelloClient {
   public static void main(String[] args) {
```

```
        try {
            // адреси RMI Registry і ім 'я сервісу
            String name = "rmi://localhost/HelloService";
            // отримуємо клієнтську заглушку
            IHelloWorld hw(IHelloWorld) Naming.lookup(name);
            // викликаємо метод віддаленого об'єкту
            System.out.println(hw.WorHello("World"));
        } catch(Exception e) {
            System.err.println("ComputePi exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

6) Запускаємо сервер.
7) Запускаємо клієнт.

1.3 Завдання на самостійну розробку

З використанням технології Java RMI реалізувати калькулятор, що підтримує базові операції (+ *, /). Обчислення повинні проводитися на віддаленому сервері (не на одній віртуальній машині!). Клієнтська частина з графічним інтерфейсом, або інтерактивним консольним, де після запуску користувач вводить необхідну операцію і операнди.

1.4 Зміст звіту

У звіті мають бути присутніми обов'язкові розділи, які вимагають правила оформлення звіту по лабораторній роботі:

- мета роботи;
- завдання на лабораторну роботу;
- короткий виклад ходу виконання роботи;
- висновок.

У додаток до звіту включити:

- роздрук коду серверної частини системи;
- роздрук коду клієнтської частини системи;
- результати роботи розподіленої системи між віддаленими JVM.

1.5 Контрольні питання

1) Що таке «Розподілені системи» і які вони мають характеристики?
2) Що таке «Інтерфейс виклику віддалених методів» і які він має характеристиками?
3) Що таке «Розподілений об'єкт», «Віддалений об'єкт»?
4) Які існують способи створення розподілених об'єктів у рамках моделі віддаленого виклику методів?
5) Який порядок дій клієнт-серверної взаємодії у рамках моделі Java RMI?

Лабораторна робота №2
ВИВЧЕННЯ СПОСОБІВ ПЕРЕДАЧІ ОБ'ЄКТІВ В RMI

Способи передачі різних об'єктів в RMI. Залежність підвищення продуктивності від кількості обчислювальних вузлів при рішенні слабозв' язаної задачі.

2.1 Короткі теоретичні відомості

Передача об'єктів в RMI. При передачі або поверненні значень з віддалених методів, аргументи можуть бути практично будь-якого типу, включаючи локальні об'єкти, віддалені об'єкти і примітивні типи. Віддаленому методу може бути переданий екземпляр будь-якого типу, який може бути примітивним типом даних, віддаленим об'єктом або серіалізуємим об'єктом, тобто тим, що реалізує інтерфейс java.io.Serializable.

Деякі типи об'єктів не задовольняють таким критеріям і тому не можуть бути передані або повернені з віддаленого методу. Більшість таких об'єктів таких, наприклад, як файловий дескриптор, інкапсулюють інформацію, яка має сенс тільки усередині одного адресного простору. Багато центральних (core) класів, включаючи класи бібліотеки java.lang and java.util, реалізують інтерфейс Serializable.

Правила, що регулюють передачу аргументів і повернення значень, наступні.

– Віддалені об'єкти передаються виключно по посиланню (by reference). Об'єктним посиланням віддаленого об'єкту remote object reference є stub, тобто "проксі" з боку клієнта, що реалізує повний набір віддалених інтерфейсів, які реалізує віддалений об'єкт (іноді stub називають також заглушкою віддаленого об'єкту).

– Локальні об'єкти передаються копіюванням, використовуючи механізм серіалізації об'єктів. За замовчанням копіюються усі поля, за винятком тих, які відмічені як static або transient. Прийнятий за замовчанням характер серіалізації може бути перевизначений.

Передача об'єкту по посиланню (як це робиться з віддаленими об'єктами) означає, що будь-які зміни, проведені над станом об'єкту при викликах віддалених методів, відбиваються в оригіналі віддаленого об'єкту. При передачі віддаленого об'єкту для того, що приймає, доступні тільки ті інтерфейси, які є віддаленими інтерфейсами. Будь-які методи, визначені в реалізації класу або визначені як невіддалені інтерфейси, що реалізовуються цим класом, для того, що приймає, недоступні.

У віддалених методах об'єкти (параметри, що повертають значення і виключення), які не є віддаленими об'єктами, передаються не по посиланню, а за значенням. Це означає, що на приймаючій віртуальній машині створюється копія об'єкту. Будь-які зміни в їх стані, виконані на стороні того, що приймає, відбиваються тільки на цій клієнтській копії, але не на початковому екземплярі.

2.2 Хід роботи

Створення демонстраційного додатку.

Обчислювальний сервер дозволяє виконувати призначені для користувача завдання. Клієнт передає серверу об’єкт, що дозволяє обчислювати число Pi із заданою точністю. Сервер повертає результат обчислення.

1) Інтерфейс сервера.

```
package compute;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {
    //всього один метод, для виконання завдання
   Object executeTask(Task t) throws RemoteException;
}
```

2) Інтерфейс задачі, що передається серверу. Інтерфейс Task визначає один метод execute, який повертає Object, не має параметрів і не видає виключень. Оскільки інтерфейс не розширює Remote, метод в цьому інтерфейсі не повинний перераховувати java.rmi.RemoteException у своїй умові throws. Інтерфейс Task розширює інтерфейс java.io.Serializable. RMI використовує механізм серіалізації об’єктів для передачі об’єктів по значеннях між віртуальними машинами Java. Реалізація Serializable відмічає клас як здатний до перетворення в потік байтів та самовизначення (self describing byte stream), який при зворотному читанні об’єкту з потоку можна реконструювати як точну копію серіалізованого об’єкту.

```
package compute;
import java.io.Serializable;

public interface Task extends Serializable {
   Object execute();
}
```

3) Реалізація сервера.

```
package engine;
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import compute.*;

public class ComputeEngine extends UnicastRemoteObject implements
Compute {
        public ComputeEngine() throws RemoteException {
        super();
```

```
    }

    public Object executeTask(Task t) {
       return t.execute();
    }

    public static void main(String[] args) {
        System.setProperty("java.security.policy", "D:\rmi.policy");
       if(System.getSecurityManager() == null) {
          System.setSecurityManager(new RMISecurityManager());
       }
       String name = "rmi://localhost/Compute";
       try {
         LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
          Compute engine = new ComputeEngine();
          Naming.rebind(name, engine);
          System.out.println("ComputeEngine bound");
       } catch(Exception e) {
          System.err.println("ComputeEngine exception: " + e.getMessage());
          e.printStackTrace();
       }
    }
  }
```

4) Створюємо файл rmi.policy. Шлях до файлу повинний співпадати з вказаним в класі сервера. Вміст файлу:

```
    grant {
         permission java.net.SocketPermission "*:1024-65535", "connect,
accept";
         permission java.io.FilePermission "", "read";
         permission java.net.SocketPermission "*:80", "connect";
    };
```

5) Реалізуємо задачу.

```
    package client;
    import compute.*;
    import java.math.*;

    // Обчислюємо число Пі. Алгоритм нас не сильно цікавить, головне, що
реалізуємо інтерфейс Task
    public class Pi implements Task {
```

```
        /** constants used in pi computation */
        private static final BigDecimal ZERO = BigDecimal.valueOf(0);
        private static final BigDecimal  ONE = BigDecimal.valueOf(1);
        private static final BigDecimal FOUR = BigDecimal.valueOf(4);

        /** rounding mode to use during pi computation */
        private static final int roundingMode =
BigDecimal.ROUND_HALF_EVEN;

        /** digits of precision after the decimal point */
        private int digits;

        /** Construct a task to calculate pi to the specified precision */
        public Pi(int digits) {
           this.digits = digits;
        }

        /** Calculate pi  */
        public Object execute(){
           return computePi(digits);
        }

        /** Compute the value of pi to the specified number of digits after the
decimal point.  The value is
         * computed using Machin's formula:
         *        pi/4 = 4*arctan  arctan(1/239)
         * and a power series expansion of arctan(x) to sufficient precision.
         */
        public static BigDecimal computePi(int digits) {
           int scale = digits + 5;
           BigDecimal arctan1_5 = arctan(5, scale);
           BigDecimal arctan1_239 = arctan(239, scale);
           BigDecimal pi =
arctan1_5.multiply(FOUR).subtract(arctan1_239).multiply(FOUR);
           return pi.setScale(digits, BigDecimal.ROUND_HALF_UP);
        }
        /** Compute the value, in radians, of the arctangent of  the inverse of the
supplied integer to the speficied
```

```
        * number of digits after the decimal point.  The valueis computed using
the power series expansion for
        * the arc tangent:
        * arctan = x /3/5 /7 +  (x^9)/9 ...
        */
       public static BigDecimal arctan(int inverseX, int scale)
       {
          BigDecimal result, numer, term;
          BigDecimal invX = BigDecimal.valueOf(inverseX);
          BigDecimal invX2 = BigDecimal.valueOf(inverseX * inverseX);
          numer = ONE.divide(invX, scale, roundingMode);
          result = numer;
          int i = 1;
          do {
             numer =
                numer.divide(invX2, scale, roundingMode);
             int denom = 2 * i + 1;
             term = numer.divide(BigDecimal.valueOf(denom) scale,
roundingMode);
             if((i % 2) != 0) {
                result = result.subtract(term);
             } else {
                result = result.add(term);
             }
             i++;
          } while(term.compareTo(ZERO) != 0);
          return result;
       }
    }
```

6) Реалізовуємо клієнт.

```
    package client;
    import java.rmi.*;
    import java.math.*;
    import compute.*;

    public class ComputePi {
       public static void main(String args[]) {
         System.setProperty("java.security.policy", "D:\\rmi.policy");
```

```
        if(System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            String name = "rmi://localhost/Compute";
            Compute comp(Compute) Naming.lookup(name);
            Pi task = new Pi(Integer.parseInt(args[0]));
            BigDecimal pi(comp.executeTask(task));
            System.out.println(pi);
        } catch(Exception e) {
            System.err.println("ComputePi exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

7) Експортуємо отримані пакети в jar файли. Отримуємо client.jar, compute.jar, engine.jar. Розміщуємо client.jar і engine.jar на різних комп'ютерах або на одному комп' ютері в різних теках(слід врахувати, що в початковому тексті клієнта прописань сервер rmiregistry). compute.jar повинний знаходиться і на клієнтові і на сервері.

8) Запускаємо сервер:

```
java  Djava.rmi.server.useCodebaseOnly=false  cp "compute.jar;engine.jar"
engine.ComputeEngine
```

9) Запускаємо клієнт:

```
java  Djava.rmi.server.codebase="file:///d:/lab2_client.jar"  cp "
compute.jar;client.jar" client.ComputePi 20
```

Параметр java.rmi.server.codebase визначає місце, звідки сервер зможе завантажити клас задачі. Оскільки при серіалізації здійснюється серіалізація його стану, але сам клас не передається можна вказати або шлях до файлу або URL. Відповідні дозволи мають бути прописані в rmi.policy.

2.3 Завдання на самостійну розробку

Реалізувати програмну систему паралельного і розподіленого відновлення паролів.

Опис системи.

Існує декілька обчислювальних вузлів. Клієнт підключається до декількох вузлів і передає їм задачі, що здійснюють підбір пароля по вказаному значенню md5. Задача отримує діапазон рядків для перевірки і md5 пароля, перевіряє вказаний діапазон рядків символів на предмет збігу md5 рядка з еталонним md5. Наприклад: діапазон початок "aaaaa" кінець "zzzzz", md5="aeaeaeaeaeaeaeae". У

якості допустимого набору символів прийняти маленькі англійські букви. Задача повертає співпадаючий рядок, якщо він знайдений.

Визначити залежність швидкодії від кількості використовуваних обчислювальних вузлів.

2.4 Зміст звіту

У звіті мають бути присутніми обов'язкові розділи, які вимагають правила оформлення звіту по лабораторній роботі:

- мета роботи;
- завдання на лабораторну роботу;
- короткий виклад ходу виконання роботи;
- висновок.

У додаток до звіту включити:

- роздрук коду серверної частини системи;
- роздрук коду клієнтської частини системи;
- файл rmi.policy;
- результати роботи розподіленої системи між віддаленими JVM;
- графік залежності швидкодії від кількості використовуваних обчислювальних вузлів (не менше чотирьох).

2.5 Контрольні питання

1) Які існують технології масштабування розподілених систем?
2) Назвіть основні завдання побудови паралельних і розподілених систем.
3) Які дії заступника клієнта і серверної заглушки у рамках моделі Java RMI?
4) Дайте визначення поняття «збережений об'єкт», «нерезидентний об’єкт».
5) Які існують способи створення розподілених об’єктів у рамках моделі Java RMI?
6) Що таке «іменування об’єкту», «прив’язка клієнта до розподіленого об’єкту», «сервер об'єктів» у рамках моделі Java RMI?
7) Які можливі звернення сервера до об’єкту і види прив'язки клієнта до об’єкту у рамках моделі Java RMI?
8) Які існують способи передачу параметрів в одній JVM і між віддаленими JVM у рамках моделі Java RMI?
9) Що таке «Серіалізація об'єкту»?
10) Які характеристики має автоматична серіалізація, серіалізація вручну, і не серіалізуємі об’єкти у рамках моделі Java RMI?

Лабораторна робота №3
ДОСЛІДЖЕННЯ ВПЛИВУ КОМУНІКАЦІЙ МІЖ ПІДЗАДАЧАМИ НА ВЕЛИЧИНУ НАКЛАДНИХ ВИТРАТ

Етапи розробки паралельного та розподіленого алгоритму обчислень, показники ефективності розробленого алгоритму. Залежність підвищення продуктивності і вплив комунікацій між підзадачами на величину накладних витрат від кількості обчислювальних вузлів при рішенні слабозв'язаної задачі.

3.1 Короткі теоретичні відомості

3.1.1 Показники ефективності паралельного алгоритму

Прискорення паралельного і розподіленого алгоритму є його найбільш інформативною характеристикою, яка показує, в скільки разів застосування паралельного алгоритму зменшує година рішення задачі в порівнянні з послідовним алгоритмом.

Прискорення паралельного і розподіленого алгоритму визначається величиною:

$$Sp = T1/Tp(n), \quad (3.1)$$

де T1 – час виконання алгоритму на одному узлі,

Tp – час виконання алгоритму на p узлах,

n – складність задачі.

Ідеальним, очевидно, являється прискорення P. У реальності це прискорення недосяжне. Перерахуємо причини неможливості досягнення ідеального прискорення:

– відсутність максимального паралелізму в алгоритмі;

– незбалансованість завантаження узлів (якщо, наприклад, необхідно скласти два вектори з 9 чисел кожен на восьмипроцесорній системі);

– тимчасові витрати на обмін даними, конфлікти, на синхронізацію.

Ефективність використання паралельним алгоритмом процесорів при рішенні задачі визначається співвідношенням :

$$Ep=T1= Sp/ P \quad (3.2)$$

Величина ефективності визначає середню долю годині виконання алгоритму, впродовж якої процесори реально використовуються для вирішення завдання.

Можна виділити наступні головні заподій втрати ефективності паралельних і розподілених обчислень:

– час ініціалізації паралельної і розподіленої програми;

– незбалансованість завантаження узлів;

– витрати на комунікації;

– наявність в програмі послідовних частин.

Час ініціалізації паралельної і розподіленої програми – це час генерації початкових даних (часто на одному узлі) і розсилки результатів по усіх узлах.

Незбалансованість завантаження узлів призводить до того, що частина узлів вимушена деякий година простоювати.

Витрати на комунікації визначаються пропускною спроможністю комунікаційної мережі, латентністю комунікаційної мережі (час підготовки до передачі інформації по каналу мережі), діаметром комунікаційної мереж і(який значною мірою визначається топологією комунікаційної мережі). Комунікаційні витрати можуть бути скорочені за рахунок використання зустрічних обмінів даними і асинхронної передачі даних. Сучасні комунікаційні засоби розподілених обчислювальних систем, а також комунікаційні бібліотеки допускають використання зустрічних обмінів, коли дані передаються одночасно в обидві сторони. Асинхронні обміни даними (обміни даними, що виконуються паралельно з виконанням обчислень) також підтримуються більшістю сучасних розподілених систем, як на апаратному, так і програмному рівнях.

3.1.2 Етапи розробки паралельного та розподіленого алгоритму

Декомпозиція. На цьому етапі виконуються аналіз задачі і оцінка можливості розпаралелювання. Задача і пов'язані з нею дані розділяються на дрібніші частини підзадачі і фрагменти структур даних. Особливості архітектури конкретної розподіленої системи на цьому етапі можуть не враховуватися.

Проектування комунікацій (обмінів даними) між підзадачами. Визначаються комунікації, необхідні для пересилки початкових даних, проміжних результатів виконання підзадач, а також комунікації, необхідні для управління роботою підзадач. Вибираються методи комунікації.

Укрупнення. Підзадачі об’єднуються у більші блоки, якщо це дозволяє підвищити ефективність алгоритму і понизити трудомісткість розробки.

Планування обчислень. Розподіл підзадач між узлами. Основний критерій вибору способу розміщення підзадач – ефективне використання узлів з мінімальними витратами часу на обміни даними.

Розглянемо ці етапи детальніше.

Технологія підготовки паралельних і розподілених додатків у вигляді наступних етапів:

1. Декомпозиція задачі на підзадачі, які реалізуються незалежно.

2. Визначення для сформованого набору підзадач інформаційних взаємодій.

3. Масштабування підзадач, визначення кількості узлів.

4. Визначення архітектури системи, закріплення підзадач за узлами, складання розкладу.

Після виконання вказаних етапів і оцінки якості паралельного та розподіленого алгоритму (прискорення, ефективності, масштабованості) може виявитися необхідним повторення деяких (чи усіх) етапів. Якщо в результаті ряду спроб бажані показники якості не досягаються, слід проаналізувати і, можливо, змінити математичну постановку задачі з метою побудови нової обчислювальної схеми.

Слід зауважити, що вказана послідовність етапів носить умовний характер. Часто, приступаючи до розробки паралельного та розподіленого алгоритму, користувач орієнтується на конкретну обчислювальну систему, зокрема, може бути відоме можливе число доступних узлів. Ясно, що на етапі декомпозиції за даними слід використати цю інформацію для вибору числа областей, що визначають число підзадач.

Якщо точне число узлів невідоме, але задані межі доступного вирішального поля, можна розпочати з масштабування базового набору завдань, а потім виконати декомпозицію і виявлення зв'язків за інформацією. Іншими словами, в приведеній загальній схемі необхідним є лише зміст етапів, тоді як самі етапи можуть виконуватися у будь-якій послідовності, притому будь-який з них може виявитися як початковим, так і завершуючим.

3.2 Завдання на самостійну розробку

Реалізувати розподілену симуляцію гри Конвея «Життя».

Дослідити залежність накладних витрат від кількості серверів і складності вирішуваної задачі.

Правила гри Конвея «Життя»

Дія гри відбувається на деякій площині, розділеній на клітини. Кожна клітина оточена 8 такими ж клітинами (околиця Мура). Кожна клітина може знаходитися в двох станах живому або мертвому, тобто порожньому. На стан будь-якої клітини роблять вплив стан сусідніх клітин. У часі ці стани дискретно відповідно до деяких правив або ГЕНЕТИЧНИХ ЗАКОНІВ КОНВЕЯ, що складаються з 2 пунктів:

1) ВИЖИВАННЯ АБО ЗАГИБЕЛЬ. Якщо жива клітина має менше 2 або більше 3 сусідів в околиці з 8 клітин ті в наступному поколінні вона помирає (моделювання реальних умів нестачі харчування або перенаселеності), інакше вона виживає;

2) НАРОДЖЕННЯ. У порожній клітині з'являється жива клітина, якщо у початкової клітини рівно 3 сусіди.

Загибель і народження усіх організмів відбувається одночасно.

Ситуації, що виникають в грі, дуже схожі на реальні процеси, що відбуваються при зародженні, розвитку і загибелі живих організмів.

Гра "Життя" належить до категорії так званих моделюючих ігор – ігор, які в тому або іншому ступені імітують процеси, що відбуваються в реальному житті.

Основна ідея гри полягає в тому, щоб, почавши з якого-небудь простого розташування живих клітин, простежити за еволюцією вихідної позиції під дією вищезгаданих ГЕНЕТИЧНИХ ЗАКОНІВ КОНВЕЯ, які управляють народженням, загибеллю і виживанням клітин. ГЕНЕТИЧНІ ЗАКОНИ КОНВЕЯ задовольняють трьом основним умовам :

1) не повинне бути жодної початкової конфігурації, для якої існував би простий доказ можливості необмеженого зростання популяції;

2) повинні існувати такі початкові конфігурації, які свідомо мають здатність безмежно розвиватися;

3) повинні існувати прості початкові конфігурації, які за значний проміжок часу ростуть, зазнають різноманітні зміни і закінчують свою еволюцію одним з наступних трьох способів:

а) повністю зникають;

б) переходять в стійку конфігурацію і перестають змінюватися взагалі;

в) виходять на коливальний режим з певним періодом.

Іноді колонія клітин поступово вимирає, проте статися це може не відразу, а лише після того, як зміниться дуже багато поколінь. У більшості своїй початкові конфігурації або переходять в стійкі і перестають змінюватися, або назавжди переходять в коливальний режим. При цьому, конфігурації, що не мали на початку гри симетрії, виявляють тенденцію до переходу в симетричні форми. Набуті властивості симетрії в процесі еволюції не втрачаються, а симетрія конфігурації може лише збагачуватися.

3.3 Зміст звіту

У звіті мають бути присутніми обов'язкові розділи, які вимагають правила оформлення звіту по лабораторній роботі:

- мета роботи;
- завдання на лабораторну роботу;
- короткий виклад ходу виконання роботи;
- висновок.

У додаток до звіту включити:

- роздрук коду серверної частини системи;
- роздрук коду клієнтської частини системи;
- результати роботи розподіленої системи між віддаленими машинами;
- графік залежності показників ефективності від кількості використовуваних обчислювальних вузлів (не менше чотирьох) для двох різних наборів даних.

3.4 Контрольні питання

1) Які існують етапи розробки паралельного і розподіленого алгоритму?
2) Якими показниками можна охарактеризувати ефективність розробленого паралельного і розподіленого алгоритму?
3) Що таке «надлінійне прискорення»?
4) Назвіть причини появи «надлінійного прискорення».
5) Опишіть перший закон Амдаля.
6) Опишіть другий закон Амдаля.
7) Опишіть третій закон Амдаля.
8) Охарактеризуйте «ефект Амдаля».
9) Як впливають накладні витрати на показники ефективності паралельного і розподіленого алгоритму?
10) Що таке «Функція ізоефективності»?

Лабораторна робота №4
ПРОГРАМНІ ЗАСОБИ ОРГАНІЗАЦІЇ КЛАСТЕРНИХ СИСТЕМ. ІНТЕРФЕЙС MPI

Способи організації кластерних систем. Залежність показників ефективності від кількості і виду обчислювальних вузлів при рішенні слабозв'язаної задачі.

4.1 Короткі теоретичні відомості

4.1.1 Кластерні системи

Кластер – це різновид паралельної або розподіленої системи, яка складається з декількох пов'язаних між собою комп'ютерів і використовується як єдиний, уніфікований комп'ютерний ресурс.

При побудові кластерів можна виділити два наступні підходи:

– у кластерну систему збираються усі доступні комп'ютери, які також можуть функціонувати і окремо. Наприклад, в таку кластерну систему можна об’єднати комп'ютери, що знаходяться в учбовій аудиторії або підключені до університетської мережі;

– у кластерну систему цілеспрямовано з'єднуються ЕОМ, що промислово випускаються. При цьому створюється потужний обчислювальний ресурс. Цей підхід дозволяє здешевити саму кластерну систему, оскільки не вимагається забезпечувати кожен окремий вузол монітором, клавіатурою і іншими периферійними прибудовами.

4.1.2 Класифікація кластерів

1) Кластери високої доступності.

Створюються для забезпечення високої доступності сервісу, що надається кластером. Надмірне число вузлів, що входять в кластер, гарантує надання сервісу у разі відмови одного або декількох серверів.

2) Кластери розподілу навантаження.

Принцип їх дії будується на розподілі запитів через один або декілька вхідних вузлів, які перенаправляють їх на обробку в інші, обчислювальні вузли. Первинна мета такого кластера – продуктивність, проте, в них часто використовуються також і методи, що підвищують надійність. Подібні конструкції називаються серверними фермами.

3) Обчислювальні кластери.

Кластери використовуються в обчислювальних цілях, зокрема в наукових дослідженнях. Для обчислювальних кластерів істотними показниками є висока продуктивність процесора в операціях над числами з плаваючою точкою (flops) і низька латентність об'єднуючої мережі, і менш істотними швидкість операцій введення-виведення, яка більшою мірою важлива для баз даних і web-сервісів. Обчислювальні кластери дозволяють зменшити час розрахунків, в порівнянні з одним комп'ютером, розбиваючи задачі на підзадачі, що паралельно виконуються, які обмінюються даними по зв'язуючій мережі.

4) Системи розподілених обчислень (grid).

Такі системи не прийнято вважати кластерами, але їх принципи значною мірою схожі з кластерною технологією. Їх також називають grid системами. Головна відмінність – низька доступність залученого вузла, тобто неможливість гарантувати його роботу в певний момент часу (вузли підключаються і відключаються в процесі роботи), тому задача має бути розбита на ряд незалежних один від одного процесів. Така система, на відміну від кластерів, не схожа на єдиний комп'ютер, а служить спрощеним засобом розподілу обчислень. Нестабільність конфігурації, у такому разі, компенсується більшим числом вузлів.

5) Кластер серверів, що організовані програмно.

Кластер серверів – група серверів, об’єднаних логічно, здатних обробляти ідентичні запити і що використовуються як єдиний ресурс. Найчастіше сервери групуються за допомогою локальної мережі. Група серверів має більшу надійність і більшу продуктивність, ніж один сервер. Об’єднання серверів в один ресурс відбувається на рівні програмних протоколів.

4.1.3 Інтерфейс MPI

У кластерних системах з розподіленою пам'яттю процесори працюють незалежно один від одного. Для організації паралельних обчислень в таких умовах необхідно мати можливість розподіляти обчислювальне навантаження і організувати інформаційну взаємодію (передачу даних) між процесорами.

Вирішення усіх перерахованих питань і забезпечує інтерфейс передачі даних (message passing interface – n MPI).

Під паралельною програмою у рамках MPI розуміється безліч одночасно виконуваних процесів. Процеси можуть виконуватися на різних процесорах, але на одному процесорі можуть розташовуватися і декілька процесів (у цьому випадку їх виконання здійснюється в режимі розділення часу). У граничному випадку для виконання паралельної програми може використовуватися один процесор – як правило, такий спосіб застосовується для початкової перевірки правильності паралельної програми.

Кожен процес паралельної програми породжується на основі копії одного і того ж програмного коду (модель SPMP). Цей програмний код, представлений у вигляді виконуваної програми, має бути доступний у момент запуску паралельної програми на усіх використовуваних процесорах. Початковий програмний код для виконуваної програми розробляється з використанням тієї або іншої реалізації бібліотеки MPI.

MPICH найвідоміша реалізація MPI, створена в Арагонській національній лабораторії (США). MPICH для Windows полягає із наступних компонентів:

- Менеджер процесів smpd.exe, який є системною службою (сервісне застосування). Менеджер процесів веде список обчислювальних вузлів системи, і запускає на цих вузлах MPI-програми, надаючи їм необхідну інформацію для роботи і обміну повідомленнями.
- Заголовні файли (.h) і бібліотеки стадії компіляції(.lib), необхідні для розробки MPI-программ.
- Бібліотеки часу виконання(.dll), необхідні для роботи MPI програм.

– Додаткові утиліти (.exe), необхідні для налаштування MPICH і запуску MPIпрограмм.

Усі компоненти, окрім бібліотек часу виконання, встановлюються за умовчанням в теку C :Files; dll бібліотеки встановлюються в C:.

Менеджер процесів є основним компонентом, який повинен бути встановлений і налагоджений на усіх комп'ютерах мережі (бібліотеки часу виконання можна, в крайньому випадку, копіювати разом з MPI програмою). Інші файли потрібно для розробки MPI програм і налаштування деякого «головного» комп'ютера, з якого проводитиметься їх запуск.

Менеджер працює у фоновому режимі і чекає запитів до нього з мережі із сторони «головного» менеджера процесів (за замовчанням використовується мережевий порт 8676). Щоб якось убезпечити себе від хакерів і вірусів, менеджер вимагає пароль при зверненні до нього. Коли один менеджер процесів звертається до іншому менеджера процесів, він передає йому свій пароль. Звідси слідує, що треба вказувати один і тій же пароль при установці MPICH на комп'ютери мережі.

У сучасних кластерах мережа передачі даних зазвичай відділяється від керуючої мережі.

Запуск MPI програми проводиться таким чином (рисунок 5.1).

1) Користувач з допомогою програми Mpirun (чи Mpiexec, при використанні MPICH2 під Windows) вказує ім'я виконуваного файлу MPI програми і необхідне число процесів. Крім того, можна вказати ім'я користувача і пароль: процеси MPI програми запускатимуться від імені цього користувача.

2) Mpirun передає відомості про запуску локальному менеджерові процесів, у якого є список доступних обчислювальних вузлів.

3) Менеджер процесів звертається до обчислювальних вузлів за списком, передаючи запущеним на них менеджерам процесів вказівки по запуску MPI програми.

4) Менеджери процесів запускають на обчислювальних вузлах декілька копій MPI програми (можливо, по декілька копій на кожному вузлі), передаючи програмам необхідну інформацію для зв'язку один з одним.

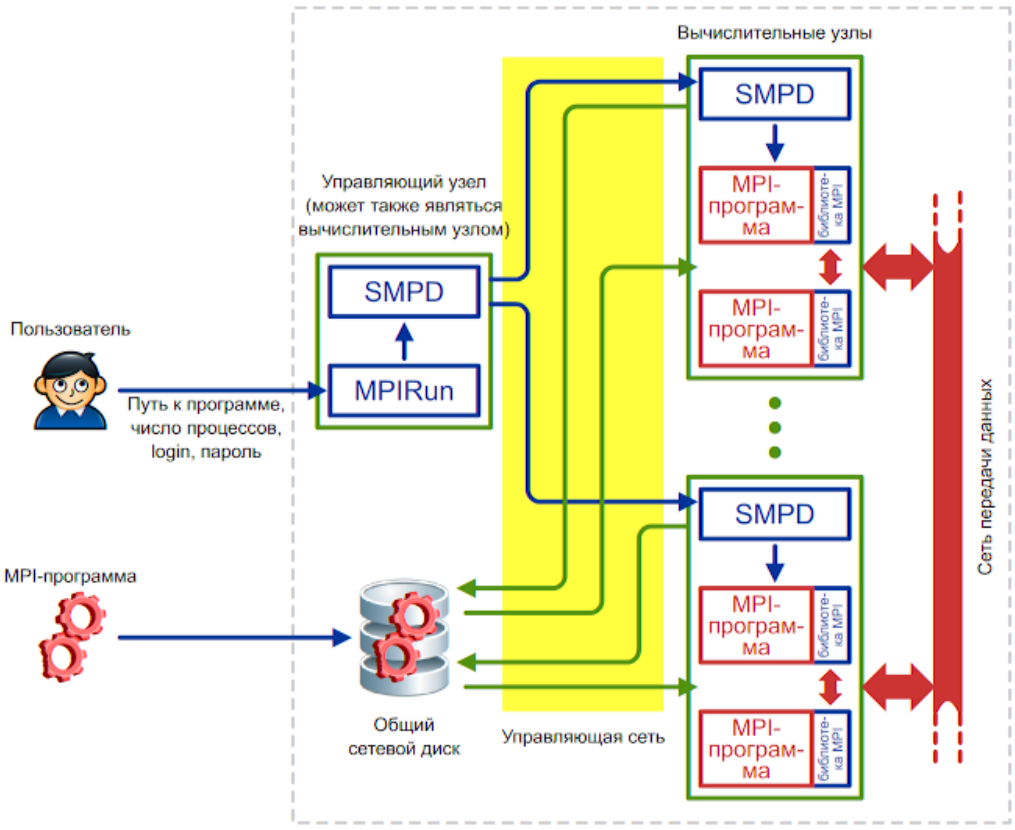

Рисунок 5.1 – Схема роботи MPICH на кластері

Дуже важливим моментом тут є те, що перед запуском MPI програма не копіюється автоматично на обчислювальні вузли кластера. Замість цього менеджер процесів передає вузлам шлях до виконуваного файлу програми точно в тому виді, в якому користувач вказав цей шлях програмі Mpirun. Це означає, що якщо ви, наприклад, запускаєте програму C:\proga.exe, то усі менеджери процесів на обчислювальних вузлах намагатимуться запустити файл C:\proga.exe. Якщо хоча б на одному з вузлів такого файлу не виявиться, станеться помилка запуску MPI програми.

Щоб кожен раз не копіювати вручну програму і усі необхідні для її роботи файли на обчислювальні вузли кластера, зазвичай використовують загальний мережевий ресурс. У цьому випадку користувач копіює програму і додаткові файли на мережевий ресурс, видимий усіма вузлами кластера, і вказує шлях до файлу програми на цьому ресурсі. Додатковою зручністю такого підходу є ті, що за наявності можливості запису на загальний мережевий ресурс запущені копії програми можуть записувати туди результати своєї роботи.

Робота MPI програми відбувається таким чином.

1) Програма запускається і ініціалізує бібліотеку часу виконання MPICH шляхом виклику функції MPI_Init.

2) Бібліотека отримує від менеджера процесів інформацію про кількості і місцеположення інших процесів програми, і встановлює з ними зв'язок.

3) Після цього запущені копії програми можуть обмінюватися один з одним інформацією за допомогою бібліотеки MPICH. З точки зору операційної системи бібліотека є частиною програми (працює в тому ж процесі), тому мож-

на вважати, що запущені копії MPI програми обмінюються даними безпосередньо один з одним, як будь-які інші додатки, що передають дані по мережі.

4) Консольний ввід-вивід усіх процесів MPI програми перенаправляється на консоль, на якій запущена Mpirun. Перенаправленням ввода-виводу займаються менеджери процесів, оскільки саме вони запустили копії MPI програми, і тому можуть отримати доступ потокам вводу-виводу програм.

5) Перед завершенням усі процеси викликають функцію MPI_Finalize, яка коректно завершує передачу і прийом усіх повідомлень, і відключає MPICH.

Кількість процесів і число використовуваних процесорів визначається у момент запуску паралельної програми засобами середовища виконання MPI програм і в ході обчислень мінятися не може. Усі процеси програми послідовно перенумеровані від 0 до np-1, де np є загальна кількість процесів. Номер процесу іменується рангом процесу.

Операції передачі даних.

Основу MPI складають операції передачі повідомлень. Серед передбачених у складі MPI функцій розрізняються парні (point to point) операції між двома процесами і колективні (collective) комунікаційні дії для одночасної взаємодії декількох процесів.

Для виконання парних операцій можуть використовуватися різні режими передачі, серед яких синхронний, блокуючий та ін.

Як вже відзначалося раніше, стандарт MPI передбачає необхідність реалізації більшості основних колективних операцій передачі даних.

Поняття комунікаторів.

Процеси паралельної програми об’єднуються в групи. Під комунікатором в MPI розуміється спеціально створюваний службовий об’єкт, що об’єднує у своєму складі групу процесів і ряд додаткових параметрів (контекст), використовуваних при виконанні операцій передачі даних.

Як правило, парні операції передачі даних виконуються для процесів, що належать одному і тому ж комунікаторові. Колективні операції застосовуються одночасно для усіх процесів комунікатора. Як результат, вказівка використовуваного комунікатора є обов'язковою для операцій передачі даних в MPI.

У ході обчислень можуть створюватися нові і віддалятися існуючі групи процесів і комунікатори. Один і тій же процес може належати різним групам і комунікаторам. Усі наявні в паралельній програмі процеси входять до складу створюваного за замовчанням комунікатора з ідентифікатором MPI_COMM_WORLD.

При необхідності передачі даних між процесами з різних груп необхідно створювати глобального комунікатора (intercommunicator).

Типи даних.

При виконанні операцій передачі повідомлень для вказівки передаваних або отримуваних даних у функціях MPI необхідно вказувати тип даних, що пересилаються. MPI містить великий набір базових типів даних, багато в чому співпадаючих з типами даних в алгоритмічних мовах. Крім того, в MPI є можливості для створення нових похідних типів даних для точнішого і коротшого опису вмісту повідомлень, що пересилаються.

Віртуальні топології.

Як вже відзначалося раніше, парні операції передачі даних можуть бути виконані між будь-якими процесами одного і того ж комунікатора, а в колективній операції беруть доля усі процеси комунікатора. У цьому плані, логічна топологія ліній зв'язку між процесами має структуру повного графа (незалежно від наявності реальних фізичних каналів зв'язку між процесорами).

Разом з цим, для викладу і подальшого аналізу ряду паралельних алгоритмів доцільне логічне представлення наявної комунікаційної мережі у вигляді тихий або інших топологій.

У MPI є можливість представлення безлічі процесів у вигляді решітки довільної розмірності. При цьому, граничні процеси решітки можуть бути оголошені сусідніми і, тим самим, на основі решітки можуть бути визначені структури типу тор.

Крім того, в MPI є засоби і для формування логічних (віртуальних) топологій будь-якого необхідного типу.

4.2 Хід роботи

У подальшому припустимо, що є декілька обчислювальних вузлів, працюючих під управлінням Windows.

1) Установка MPICH в Windows

Спочатку треба викачати останню версію MPICH з цієї сторінки: http://www.mpich.org/downloads/.

Завантажений інсталятор необхідно *запустити з привілеями адміністратора* на усіх комп'ютерах, на яких планується запускати MPI програми.

Під час установки треба буде ввести пароль для доступу до менеджера процесів SMPD. Необхіодно ввести однаковий пароль на усіх комп'ютерах.

Якщо Windows запитає, чи дозволити доступ в мережу програмі smpd.exe, то натисніть «Дозволити».

Тепер, швидше за все, MPICH правильно встановлений на комп'ютер. Проте, перш ніж переходити до налаштуванню, обов'язково слід перевірити дві речі: чи запущена служба «MPICH Process Manager», і чи дозволений цій службі доступ в мережу. Необхідно подивитись в списку дозволених програм «Process launcher for MPICH applications» і «Process manager service for MPICH applications».

Якщо якась з перерахованих програм відсутня в списку дозволених програм, то можна додати її вручну:

− якщо відсутній «Process launcher for MPICH applications» додайте C:\program files\mpich2\bin\mpiexec.exe;

− якщо відсутній «Process manager service for MPICH2 applications» додайте C:\program files\mpich2\bin\smpd.exe.

2) Налаштування MPICH

Розглянемо налаштування MPICH на прикладі конфігурації з двох комп'ютерів, об'єднаних в локальний мережі (Wi-Fi): один комп'ютер має мережеве

ім'я MrBig і IP-адресу 192.168.1.4, інший – ім'я Small і адресу 192.168.1.3. Припустимо, що MPI програми будемо запускати з комп'ютера MrBIG. Кожен комп'ютер має двоядерний процесор.

Передусім, треба створити на усіх комп'ютерах користувача з однаковим ім'ям і паролем; від імені цього користувача запускатимуться MPI програми. Простіше усього це зробити, встановивши однаковий пароль користувачам Адміністратор.

Тепер нам треба настроїти менеджери процесів MPICH. Для цього запускаємо на усіх комп'ютерах програму Wmpiconfig. Якщо усі попередні кроки зроблені правильно, то в полі «version» в лівій колонці таблиці ви повинні побачити версію встановленого менеджера процесів (рисунок 5.2). Якщо менеджер процесів не встановлений, або йому закритий доступ в ятір, то ви побачите напис «MPICH not installed or unable to query the host» в одному з полів лівого стовпця.

Wmpiconfig призначена для налаштування менеджерів процесів на поточному комп'ютері і інших комп'ютерах мережі. Для цього вона під'єднується до менеджерів процесів на вибраних комп'ютерах, читає наявні у них налаштування, і повідомляє їм нові налаштування, якщо потрібно. Елементи управління програми Wmpiconfig виконують наступні дії.

- Зліва знизу є список комп'ютерів, з якими працює програма налаштування. Ім'я комп'ютера на білому фоні означає, що не було спроб зв'язатися з цим комп'ютером; зелений фон означає, що зв'язок проведений успішно; сірий фон означає, що при встановленні зв'язку виникла помилка. Видалити комп'ютер зі списку можна клавішею Del. Слідує мати на увазі, що цей список призначений тільки для зручності налаштування, і не має ніякого відношення до списку комп'ютерів, на яких буде запущена MPI програма.
- Кнопка «Get Hosts» отримує список комп'ютерів в заданому домені або робочій групі (задається у спадаючому списку «Domain»). Отриманий список замінює наявний список комп'ютерів або, якщо натиснута кнопочка «+», додає комп'ютери до поточному списку.
- Кнопка «Scan Hosts» отримує налаштування з усіх комп'ютерів списку; кнопка «Scan for Versions» отримує тільки номери версій.
- Кнопка «Get Settings» отримує поточні налаштування того комп'ютера, ім'я якого введено в поле введення «Host». При виборі комп'ютера в списку комп'ютерів його ім'я автоматично вводиться в поле «Host». Якщо натиснута кнопка «Click», то налаштування будуть отримані автоматично при виборі комп'ютера зі списку.

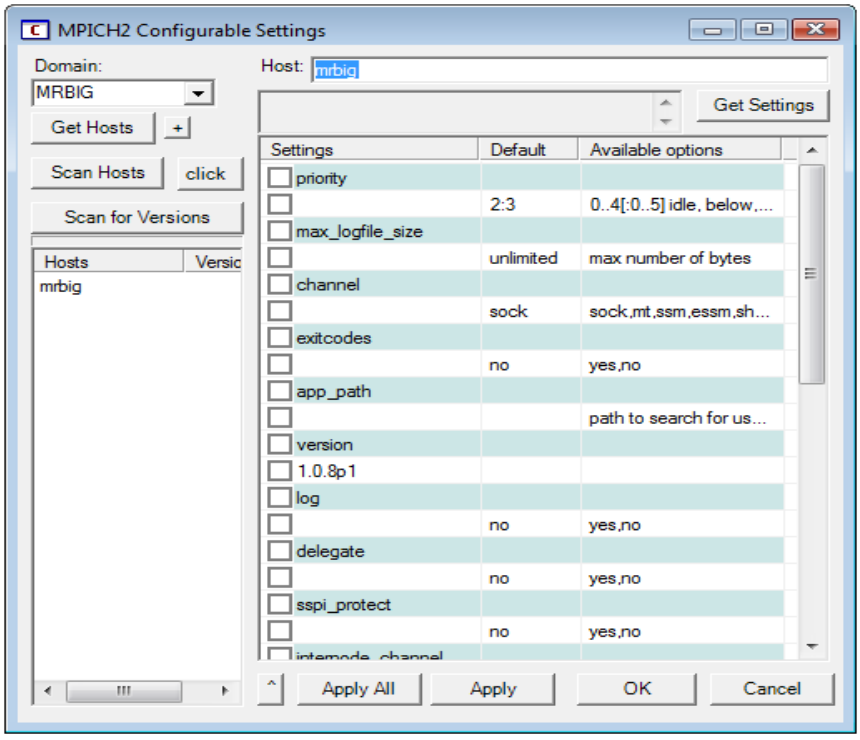

Рисунок 5.2 – Інтерфейс додатку Wmpiconfig

На ті комп'ютери, з якого планується запуск програм, треба вказати список доступних обчислювальних вузлів (рисунок 5.3). Цей список треба ввести (через пробіл) в поле hosts лівого стовпця таблиці, і натиснути кнопку «Apply». На рисунку показаний приклад, коли сам комп'ютер, з якого проводиться запуск MPI програм, являється одним з обчислювальних вузлів.

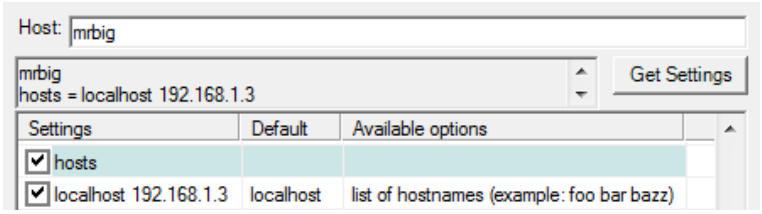

Рисунок 5.3 – Вказівка списку доступних обчислювальних вузлів

Далі потрібно перевірити, чи бачать менеджери процесів один одного по мережі. Для цього в програмі Wmpi config на «головному» комп'ютері треба ввести в поле «Host» адресу комп'ютера, що перевіряється, і натиснути «Get Settings». Необхідно побачити версію встановленого менеджера процесів на вибраному комп'ютері. Якщо зв'язок встановити не вдається – буде пауза в декілька секунд, після чого в останньому рядку таблиці з'явиться повідомлення про помилку. Якщо в попередніх налаштуваннях було все гаразд, то слід перевірити мережу: переконатися, що комп'ютери «бачать» один одного, спробувати відключити бранмауери, і тому подібне. Також перевірте, чи співпадає контрольна фраза (поле phrase) на усіх комп'ютерах.

3) Створення загального мережевого ресурсу

Для зручного запуску MPI програм слід створити на одному з комп'ютерів загальний мережевий ресурс.

− Встановлюємо «число одночасних користувачів» таким, щоб воно перевищувало кількість комп'ютерів мережі, призначених для запуску MPI програму.

− Встановлюємо властивості дозволу безпеки «Читання і виконання», «Список вмісту теки» і «Читання».

Переконаємося, що створена загальна тека видно з іншого комп'ютера. Для цього натискаємо на іншому комп'ютері Пуск → Виконати як..., і вводимо мережевий шлях до папки. У нашому випадку це \192.168.1.4\MPI

4) Створення MPI програми

Створимо просту програму, що приблизно обчислює значення числа Пі шляхом чисельного обчислення наступного інтеграла :

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx. \tag{5.1}$$

Передусім, треба налаштувати середовище розробки, щоб воно знаходило заголовні файли (./include) і .lib бібліотеки MPICH (./lib).

Тепер створимо консольний проект з початковим кодом програми.

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f(double a) {        return(4.0(1.0 + a*a));    }

int main(int argc, char *argv[])
{
  // ініціалізація і підготовка даних.
  int done = 0, n, myid, numprocs, i;
  double PI25DT = 3.14159265358979323846264 3;
  double mypi, pi, h, sum, x;
  double startwtime = 0.0, endwtime;
  int  namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  // початок паралельної частини програми
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  MPI_Get_processor_name(processor_name,&namelen);
```

```
 while(!done) {
    if(myid == 0) {
        fprintf(stdout, "Enter the number of intervals:(0 quits) ");
        fflush(stdout);
        if(scanf("%d",&n) != 1) {
                fprintf( stdout, "No number entered; quitting\n" );
         n = 0;
      }
      startwtime = MPI_Wtime();
    }

   // розсилка даних на усі процеси
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n == 0) done = 1;
    else {
        // обчислення часткової суми на кожному з процесів
        h  = 1.0-(double) n;
        sum = 0.0;
        for(i = myid + 1; i <= n; i += numprocs) {
                x = h-((double)i - 0.5);
                sum += f(x);
        }
        mypi = h * sum;

        // складання часткових сум на процесі з рангом 0
        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
        // виведення результату на процесі з рангом 0
        if(myid == 0) {
                printf("pi is approximately %.16f, Error is %.16f\n",pi, fabs);
                endwtime = MPI_Wtime();
                printf("wall clock time = %f\n", endwtime  startwtime);
                fflush( stdout );
        }
     }
    }
    MPI_Finalize();
    // кінець паралельної частини програми
    return 0;
```

}

Відкомпілюйте програму і отримаєте застосування cpi.exe.

5) Запуск MPI програм

Для запуску MPI програм в комплект MPICH входить програма з графічним інтерфейсом Wmpiexec, яка є оболонкою навколо відповідної утиліти командного рядка Mpiexec.

Вікно програми Wmpiexec показане на рисунку 5.3 (зверніть увагу, що включений прапорець «more options»).

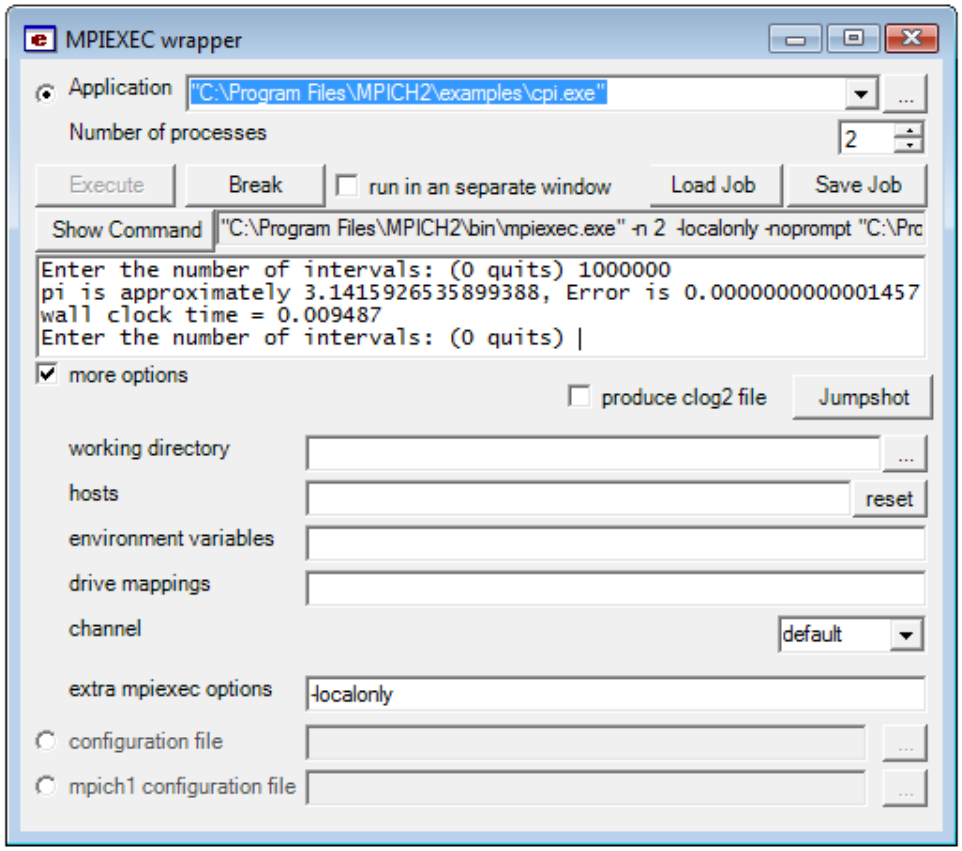

Рисунок 5.4 – Вікно програми Wmpiexec

Елементи управління вікна мають наступний сенс:

– Полі введення «Application»: сюди вводиться шлях до MPI програмі. Як вже було сказано раніше, шлях передається в незмінному вигляді на усі комп'ютери мережі, тому бажано, щоб програма розташовувалася в загальній мережевій теці.

– «Number of processes»: число процесів, що запускаються. За замовчанням процеси розподіляються порівну між комп'ютерами мережі, проте цю поведінку можна змінити за допомогою конфігураційного файлу.

– Кнопка «Execute» запускає програму; кнопка «Break» примусово завершує усі запущені екземпляри.

– Прапорець «run in a separate window» перенаправляє виведення усіх екземплярів MPI програми в окреме консольне вікно.

– Кнопка «Show Command» показує в полі справа командний рядок, який використовується для запуску MPI-програми (нагадуємо: Wmpiexec – всього лише оболонка над Mpiexec). Командний рядок збирається з усіх налаштувань, введених в інших полях вікна.

– Далі йде велике текстове поле, в яке потрапляє ввід-вивід усіх екземплярів MPI програми, якщо не встановлений прапорець «run in a separate window».

– Прапорець «more options» показує додаткові параметри.

– «working directory»: сюди можна ввести робочий каталог програми. Знову ж таки, цей шлях повинен бути вірний на усіх обчислювальних вузлах. Якщо шлях не вказаний, то як робочий каталог використовуватиметься місце знаходження MPI програми.

– «hosts»: тут можна вказати через пропуск список обчислювальних вузлів, використовуваних для запуску MPI програми. Якщо це поле порожнє, то використовується список, що зберігається в налаштуваннях менеджера процесів поточного вузла (дивіться розділ «Налаштування MPICH»).

– «environment variables«: в цьому полі можна вказати значення додаткових змінних середовища, що встановлюються на усіх вузлах на час запуску MPI програми. Синтаксис наступний: імя1 = значення1, імя2 = значення2.

– «drive mappings»: тут можна вказати мережевий диск, що підключається на кожному обчислювальному вузлі на час роботи MPI програми. Синтаксис: Z:\\winsrv\wdir.

– «channel»: дозволяє вибрати спосіб передачі даних між екземплярами MPI програми.

– «extra mpiexec options»: в це полі можна ввести додаткові ключі для командного рядка Mpiexec.

Спочатку спробуйте запустити один, два, і чотири процеси на одному комп'ютері. Щоб MPICH не розподіляв процеси, що запускалися, між наявними вузлами, відключимо роботу з мережею; для цього існує ключ командного рядка localonly. При його додаванні менеджер процесів не використовується. Це дуже корисно, якщо з будь-яких причин MPICH не вдається налаштувати. Введіть цей ключ в поле «extra mpiexec options» (рисунок 5.4).

У поле «Application» введіть шлях до програмі cpi.exe. Оскільки програма запускається на одному комп'ютері, шлях можна вводити локальний.

«Number of processes» введіть рівним 1, і натисніть кнопку «Execute». Програма запуститься і запитає число інтервалів для чисельної інтеграції. Введіть 100000000, і натисніть Ctrl+Enter. Програма порахує число Пі, і виведе час, витрачений на обчислення. Проведіть обчислення з 100 мільйонами інтервалів ще декілька разів, щоб визначити мінімальний час. Після цього введіть 0, і натисніть Ctrl+Enter. Програма завершиться. Далі запустіть програму ще декілька раз, цього разу вибравши 2 і 4 процеси.

Проаналізуйте час виконання завдання при запуску на одному вузлі.

Тепер запустіть MPI програму на двох комп'ютерах. Для цього скопіюйте програму cpi.exe на загальний мережевий диск(//192.168.1.4/MPI).

Оскільки екземпляри MPI програми працюють і обмінюються даними по мережі незалежно від менеджера процесів, їх теж треба внести в список виключень брандмауера Windows. Додайте cpi.exe (прямо з мережевого ресурсу) в список виключень брандмауера на усіх комп'ютерах мережі.

Вкажіть мережеву адресу програми cpi.exe (//192.168.1.4/MPI/cpi.exe) в полі «Application» програми Wmpiexec. Виберіть число процесів і натисніть кнопку Execute.

Проаналізуйте час виконання завдання при розподіленому запуску.

4.3 Завдання на самостійну розробку

Напишіть паралельну і розподілену програму за допомогою MPI. Зчитайте з файлу два вектори однакової довжини. Розбийте їх на приблизно однакові частини по числу процесів і розішліть їх їм колективною функцією розсилки даних. Після того, як кожен з процесів вичислить локальний скалярний добуток частин векторів, розішліть результати усім процесам відразу. Після цього нульовий процес складає отримані числа, отримуючи повний скалярний добуток; перший процес складає усі числа, окрім останнього, який навпаки, віднімає з суми; другий процес віднімає вже останні два числа і так далі. Отримані результати зберіть одним з процесів і виведіть на екран.

4.4 Зміст звіту

У звіті мають бути присутніми обов'язкові розділи, які вимагають правила оформлення звіту по лабораторній роботі:

- мета роботи;
- завдання на лабораторну роботу;
- короткий виклад ходу виконання роботи;
- висновок.

У додаток до звіту включити:

- роздрук коду програми;
- результати роботи паралельної і розподіленої програми;
- графік залежності показників ефективності від кількості використовуваних обчислювальних вузлів (не менше чотирьох) на наборах даних різної розмірності.

4.5 Контрольні питання

1) Дайте визначення поняття «кластер». Опишіть два підходи побудови кластерів.
2) Класифікуйте і опишіть кластерні системи.
3) Дайте визначення поняття «Інтерфейс передачі даних»(MPI). Опишіть його основні концепції.
4) Опишіть процес передачі і прийому повідомлень в MPI.
5) Опишіть процес передачі даних від одного процесу усім процесам в MPI.

6) Опишіть процес передачі даних від усіх процесів одному процесу в MPI.
7) Опишіть структуру вузлів мережі і ліній зв'язку в MPI.

Лабораторна робота №5
СТВОРЕННЯ МІКРОСЕРВІСІВ ЗА ДОПОМОГОЮ WSO2 MICROSERVICES FRAMEWORK ДЛЯ JAVA

Освоєння навиків роботи з WSO2 Microservices Framework для Java. Створення і реалізація мікросервісу. Обґрунтування переваг роботи з мікросервісами.

5.1 Короткі теоретичні відомості

Сервіс-орієнтована архітектура (SOA) – незалежний від технологій, компаній і програмних продуктів підхід до розробки програмного забезпечення на основі розподілених, слабо пов'язаних, замінних компонентів з чітко визначеними інтерфейсом і протоколами взаємодії. Сервіс в цій архітектурі:

- представляє бізнес-логіку з певним результатом;
- може складатися з інших сервісів або залежати від них;
- є "чорним ящиком" для своїх клієнтів;
- має свій маніфест.

Мікросервісна архітектура (MSA) – окремий випадок SOA, що спонукає будувати взаємодію наскільки це можливо і необхідно, з невеликих, слабо пов'язаних, легко змінюваних і взаємозамінних компонентів (мікросервісів). MSA орієнтована, в першу чергу, на структуру і компоненти окремого додатка.

Мікросервіс:

- виконує тільки одну досить елементарну і певну функцію (Unix way – єдина відповідальність);
- деплоїться і розробляється незалежно від інших мікросервісів;
- є незалежним від інших мікросервісів (в тому числі максимальна мінімізація загального коду);
- є "чорним ящиком" для своїх клієнтів.

Існує досить багато технологій створення та реалізації мікросервісів. В цій лабораторній роботі розглянемо особливості застосування WSO2 Microservices Framework для Java.

5.2 Хід роботи

Для створення і реалізації мікросервісу за допомогою WSO2 Microservices Framework для Java необхідно виконати чотири основні етапи.

1) Створення мікросервісу.
2) Додавання GET/POST методів до мікросервісу.
3) Додавання перехоплювачів до мікросервісу.
4) Додавання оброблювача користувацьких помилок з ExceptionMapper.

5.2.1 Створення мікросервісу

Для створення мікросервісу на PC потрібно встановити такі компоненти як: JDK (Java Development Kit), JVM (Java Virtual Machine) та Maven.

Після встановлення компонентів потрібно відкрити командну строку нашого PC і ввести нижче наведену команду:

```
mvn archetype:generate
-DarchetypeGroupId=org.wso2.msf4j
-DarchetypeArtifactId=msf4j-microservice
-DarchetypeVersion=1.0.0
-DgroupId=org.example
-DartifactId=stockquote
-Dversion=0.1-SNAPSHOT
-Dpackage=org.example.service
-DserviceClass=HelloService
```

Як тільки виконаємо вказану вище команду, вона створить директорію під назвою “stockquote” і в цьому каталозі можна знайти артефакти, пов'язані з мікросервісом:

- pom.xml – це файл проекту maven для створеного мікросервісу;
- src – цей каталог містить вихідний код для мікросервісу.

За допомогою наведеної вище команди вказуємо назву пакета як "org.example.service" і клас обслуговування як "HelloService". У каталозі src можна знайти ієрархію пакетів і вихідний файл з назвою "HelloService.java". На додаток до цього вихідного файлу, існує ще один вихідний файл з назвою “Application.java”, який буде використовуватися для запуску мікросервісу в попередньо побудованій службі.

Після створення мікросервісу можна створити проект IDE, використовуючи будь-яку з наведених нижче команд, заснованих на кращому IDE.

```
mvn idea: idea (для Intellij Idea)
mvn eclipse: eclipse (для Eclipse)
```

Як тільки це буде зроблено, можна відкрити створені вихідні файли, використовуючи бажану IDE. Автоматично згенерований вихідний файл "HelloService.java" виглядає наступним чином.

```
@Path("/service")
public class HelloService {

@GET
@Path("/")
 public String get() {
        // TODO: Implementation for HTTP GET request
        System.out.println("GET invoked");
        return "Hello from WSO2 MSF4J";
 }
}
```

Це основний клас реалізації сервісу, який буде виконувати бізнес-логіку він має конфігурації, пов'язані з мікросервісом, через анотації, які є підмножиною анотацій JAX-RS.

– @Path – ця анотація використовується для визначення базового шляху API, а також шляху до рівня ресурсів (у нас є "/ service" – як базовий шлях, а "/" – як шлях до ресурсу).

– @GET – це метод http, який додається до даного методу (для методу get()).

У вищезгаданому класі змініть контекст базового шляху з "/service" на "/stockquote", змінивши анотацію "@Path" на рівні класу.

```
@Path("/stockquote")
public class HelloService {…}
```

На додаток до цього класу, існує ще один клас, який генерується автоматично – "Application.java".

```
public class Application {
  public static void main(String[] args) {
        new MicroservicesRunner().deploy(new HelloService()).start();
  }
}
```

Цей клас використовується для запуску мікросервісу в середовищі виконання msf4j. Тут необхідно створити екземпляр класу "MicroservicesRunner" з об'єктом класу "HelloService", який реалізовує мікросервісну логіку, а потім ініціювати об'єкт викликом методу "start".

Тепер можна запустити мікросервіс, результат запуску мікросервісу можна побачити на рисунку 5.1.

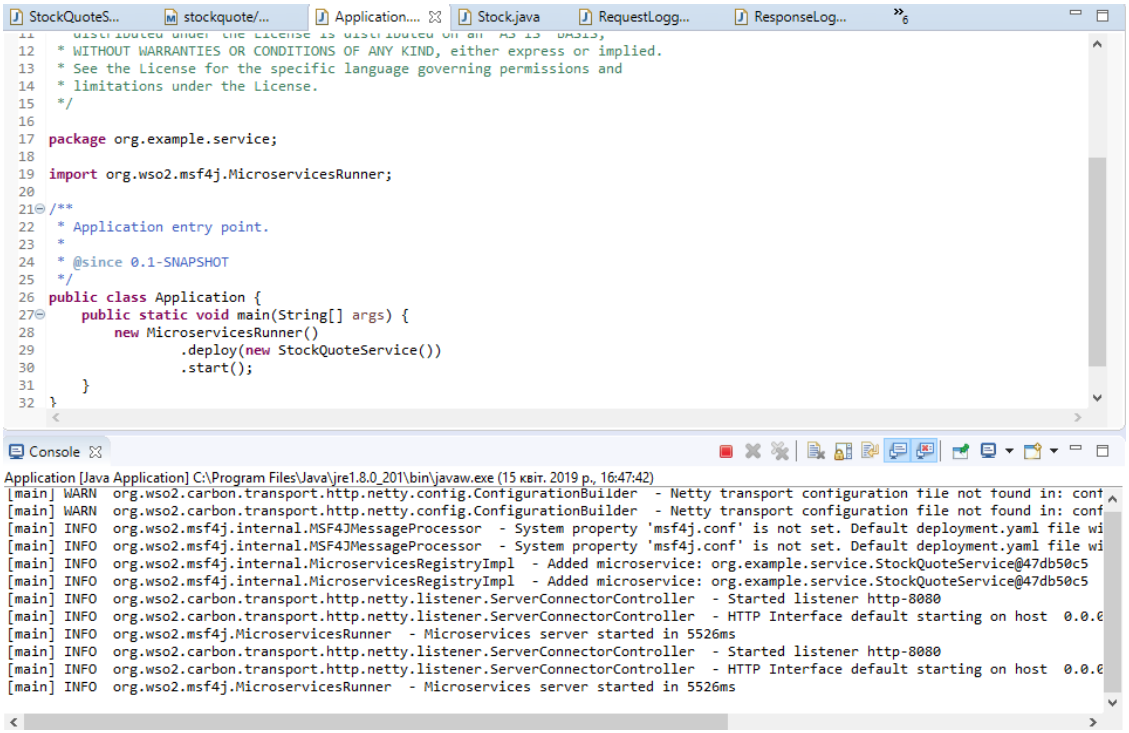

Рисунок 5.1 − Запуск мікросервісу

Тепер настав час надіслати запит і подивитися, чи справді ця служба робить те, що повинна робити (віддзеркалює повідомлення "Hello from WSO2

MSF4J"). Відкрийте інший термінал і запустіть команду curl, вказану нижче.

curl -v http://localhost:8080/stockquote

Після виконання команди отримаємо результат, який наведений на рисунку 5.2.

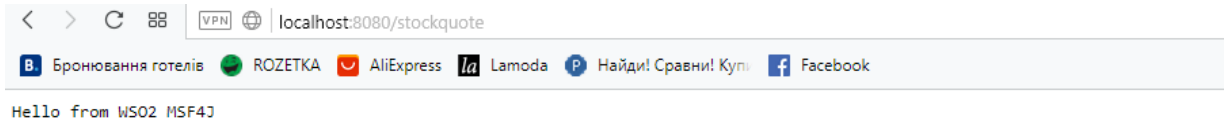

Рисунок 5.2 – Результат запиту до мікросервісу

5.2.2 Додавання GET\POST методів до мікросервісу

Наприклад, необхідно розширити згенерований клас HelloService, щоб надати інформацію про запаси на основі даних користувача. Можна перейменувати клас HelloService як "StockQuoteService" і реалізувати методи для операцій "GET" і "POST".

```
@Path("/stockquote")
public class StockQuoteService {
        private Map<String, Stock> quotes = new HashMap<>();
        public StockQuoteService() {
        quotes.put("IBM", new Stock("IBM", "IBM Inc. ", 90.87, 89.77));
}

@GET
@Path("/{symbol}")
@Produces("application/json")
public Response get(@PathParam("symbol") String symbol) {
Stock stock = quotes.get(symbol);
return stock == null ?
Response.status(Response.Status.NOT_FOUND).entity("{\"result\":\"Symbol
not found = " + symbol + "\"}").build() :
Response.status(Response.Status.OK).entity(stock).build();
}

@POST
@Consumes(“application/json”)
public Response addStock(Stock stock) {
        if(quotes.get(stock.getSymbol()) != null) {
                return Response.status(Response.Status.CONFLICT).build();
        }
        quotes.put(stock.getSymbol(), stock);
        return Response.status(Response.Status.OK).
        entity("{\"result\":\"Updated the stock with symbol = “ +
stock.getSymbol() + "\"}").build();
```

```
}
```

Тут використовуємо нову анотацію "@Produces", щоб вказати тип вмісту відповіді як "application/json". Крім того, метод get повертає об'єкт "Response", який є властивістю середовища виконання msf4j. Також використовуємо анотацію "@PathParam" для доступу до змінної параметра шляху, визначеної в анотації "@Path".

У цьому класі використаний новий клас "Stock", який містить інформацію про конкретний елемент запасу.

```
public class Stock {
   private String symbol;
   private String name;
   private double high;
   private double low;
   public Stock(String symbol, String name) {
      this.symbol = symbol;
      this.name = name
  }
   public Stock(String symbol, String name, double high, double low) {
      this.symbol = symbol;
      this.name = name;
      this.high = high;
      this.low = low;
   }
   public String getSymbol() {
      return symbol;
   }
   public void setSymbol(String symbol) {
      this.symbol = symbol;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public double getHigh() {
      return high;
   }
   public void setHigh(double high) {
      this.high = high;
   }
   public double getLow() {
      return low;
   }
```

```
    public void setLow(double low) {
        this.low = low;
    }
}
```

Тепер мікросервіс запущений і працює за замовчуванням на 8080 порту. Давайте виконаємо методи GET і POST з наступними командами.

curl -v http://localhost:8080/stockquote/IBM

Результат виконання даної команди наведено на рисунку 5.3.

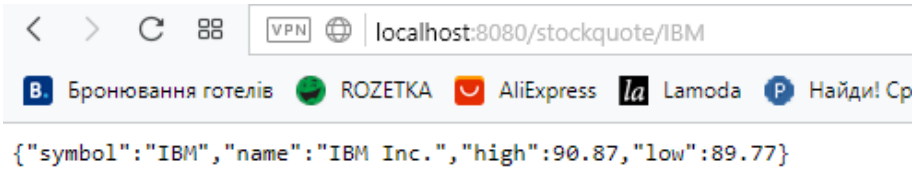

Рисунок 5.3 – Результат виконання вище наведеної команди GET

curl -v -X POST -H "Content-Type:application/json" -d ‘{"symbol":"GOOG","name":"Google Inc.", "high":190.23, "low":187.45}’ http://localhost:8080/stockquote

Результат виконання даної команди наведено на рисунку 5.4.

Рисунок 5.4 – Результат виконання вище наведеної команди POST

5.2.3 Додавання перехоплювачів до мікросервісу

Перехоплювач – це компонент, який може бути використаний для перехоплення всіх повідомлень, що надходять у конкретний мікросервіс. Перехоплювач буде отримувати доступ до повідомлення перед виконанням логіки мікросервісу. Його можна використовувати для таких цілей:

- аутентифікації;
- дроселювання;
- обмеження швидкості.

Логіка перехоплювача може бути реалізована як окрема складова з бізнес-логіки мікросервісу. Він також може бути спільним для декількох мікросервісів з моделлю, що підтримується msf4j.

Подивимося, як реалізувати логіку перехоплювача на прикладі класу RequestLoggerInterceptor.java.

```
public class RequestLoggerInterceptor implements RequestInterceptor {
  private static final Logger log = LoggerFactory.getLogger(RequestLoggerInterceptor.class);
  @Override
  public boolean interceptRequest(Request request, Response response) throws Exception {
        log.info("Logging HTTP request { HTTPMethod: {}, URI: {} }",
              request.getHttpMethod(), request.getUri());
        String propertyName = "SampleProperty";
        String property = "WSO2–2021";
        request.setProperty(propertyName, property);
        log.info("Property {} with value {} set to request", propertyName,
  property);
        return true;
  }
}
```

Наведений вище клас реалізує інтерфейс "RequestInterceptor", який надходить з середовища виконання msf4j. Він використовується для перехоплення запитів, що надходять у мікросервіс. У цьому класі він перевизначає метод "interceptRequest", який буде виконуватися після отримання запиту. Тут буде написана логіка перехоплювачів.

Аналогічно, відповідь може бути перехоплено за допомогою "ResponseLoggerInterceptor", який реалізує інтерфейс "ResponseInterceptor" середовища виконання msf4j.

Як тільки логіка перехоплювача реалізована, необхідно задіяти цей перехоплювач з мікросервісом, використовуючи анотації.

```
@Path(“/stockquote”)
public class StockQuoteService {
  private Map<String, Stock> quotes = new HashMap<>();
  public StockQuoteService() {
  quotes.put("IBM", new Stock("IBM", "IBM Inc. ", 90.87, 89.77));
}
@GET
@Path("/{symbol}")
@Produces("application/json")
@RequestInterceptor(RequestLoggerInterceptor.class)
@ResponseInterceptor(ResponseLoggerInterceptor.class)
public Response get(@PathParam("symbol") String symbol) {
  Stock stock = quotes.get(symbol);
  return stock == null ?
        Response.status(Response.Status.NOT_FOUND).build() :
```

```
        Response.status(Response.Status.OK).entity(stock).build();
    }
```

У цьому головному класі мікросервісу використані анотації "@RequestInterceptor" та "@ResponseInterceptor" для зв'язування класів перехоплювачів, які реалізують логіку.

Тепер мікросервіс запущений і працює за замовчуванням на 8080 порту. Давайте виконаємо метод GET і POST метод з наступними командами.

curl -v http://localhost:8080/stockquote/IBM
{"symbol":"IBM","name":"IBM Inc.","high":90.87,"low":89.77}

У вікні терміналу, на якому запущено мікросервіс, можна побачити наведені нижче записи журналу, які друкуються з записаних перехоплювачів.

2021–06–18 12:08:01 INFO RequestLoggerInterceptor:19 — Logging HTTP request { HTTPMethod: GET, URI: /stockquote/IBM }
2021–06–18 12:08:01 INFO RequestLoggerInterceptor:23 — Property SampleProperty with value WSO2–2021 set to request
2021–06–18 12:08:01 INFO ResponseLoggerInterceptor:18 — Logging HTTP response
2021–06–18 12:08:01 INFO ResponseLoggerInterceptor:21 — Value of property SampleProperty is WSO2–2021

5.2.4 Додавання оброблювача користувацьких помилок ExceptionMapper

Однією з основних вимог при написанні мікросервісів є можливість обробляти помилки. Цього можна досягти за допомогою концепції ExceptionMapper msf4j.

ExceptionMapper має 2 частини.

- Exception Type.
- Exception Mapper.

По-перше, користувач повинен визначити Exception Type як Exception Java.

```
public class SymbolNotFoundException extends Exception {
  public SymbolNotFoundException() {
        super();
  }
  public SymbolNotFoundException(String message) {
        super(message);
  }
  public SymbolNotFoundException(String message, Throwable cause) {
        super(message, cause);
  }
  public SymbolNotFoundException(Throwable cause) {
        super(cause);
  }
  protected SymbolNotFoundException(String message, Throwable cause,
```

```
                    boolean enableSuppression, boolean
                    writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
  }
}
```

Тоді ExceptionMapper може бути використаний для зіставлення цього виключення з заданим індивідуальним повідомленням про помилку.

```
public class SymbolNotFoundMapper
implements ExceptionMapper<SymbolNotFoundException> {
  public Response toResponse(SymbolNotFoundException ex) {
        return Response.status(404).entity(ex.getMessage() +
        " [from SymbolNotFoundMapper] ").type(“text/plain”).build();
  }
}
```

"SymbolNotFoundMapper" реалізує клас "ExceptionMapper", і в цьому випадку вказується загальний тип "SymbolNotFoundException", який відображається на цей ExceptionMapper. Усередині методу "toResponse" можна виконати індивідуальне генерування відповіді.

Тепер необхідно звернути увагу, яким чином ExceptionMapper прив'язаний до основного мікросервісу. У класі "StockQuoteService", в межах методу, який прив'язується до відповідного методу HTTP, потрібно викинути конкретний Exception, як зазначено нижче.

```
@Path("/stockquote")
public class StockQuoteService {
  private Map<String, Stock> quotes = new HashMap<>();
  public StockQuoteService() {
        quotes.put("IBM", new Stock("IBM", "IBM Inc. ", 90.87, 89.77));
  }

@GET
@Path("/{symbol}")
@Produces("application/json")
@Timed
  public Response get(@PathParam("symbol") String symbol)
  throws SymbolNotFoundException {
        Stock stock = quotes.get(symbol);
        if (stock == null) {
              throw new SymbolNotFoundException("Symbol " + symbol + "
        not found");
        }
        return Response.status(Response.Status.OK).entity(stock).build();
}
```

У вищевказаній реалізації, метод "get" – це генерація "SymbolNotFoundException", який відображається в "SymbolNotFoundMapper", про який говорилось вище. У разі виникнення помилки в межах цього методу, він відповість назад з налаштованою помилкою.

Тепер мікросервіс запущений і працює за замовчуванням на 8080 порту. Давайте виконаємо метод GET і метод з наступними командами.

```
curl -v http://localhost:8080/stockquote/WSO2
```

Результат виконання даної команди наведено на рисунку 5.5.

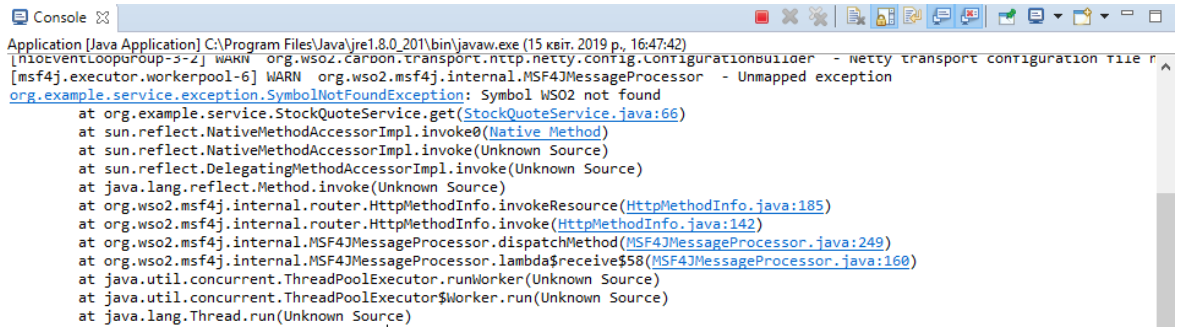

Рисунок 3.5 – Результат виконання команди приведеної вище

5.3 Завдання на самостійну розробку

1) Створіть мікросервіс відновлення пароля по вказаному значенню md5 в заданому діапазоні пошуку (див. лабораторну роботу 2) за допомогою WSO2 Microservices Framework для Java відповідно до етапів, описаних в п. 5.1:

- створення мікросервісу.
- додавання GET/POST методів до мікросервісу.
- додавання перехоплювачів до мікросервісу.
- додавання оброблювача користувацьких помилок з ExceptionMapper.

2) Створіть окремий мікросервіс, що запускає декілька екземплярів мікросервісу на різних обчислювальних вузлах.

3) Одночасно виконайте команду GET для всіх екземплярів мікросервісу, вказавши для кожного екземпляру відповідний рівномірно розподілений діапазон пошуку рядка з 4 символів англійського алфавіту.

4) Отримайте дані щодо часу виконання запиту на кожному екземплярі мікросервісу.

5) Змініть розмірність задачі, запустіть рівномірно розподілений пошук для рядка з 5 символів.

6) Отримайте дані щодо часу виконання запиту на кожному екземплярі мікросервісу.

5.4 Зміст звіту

У звіті мають бути присутніми обов'язкові розділи, які вимагають правила оформлення звіту по лабораторній роботі:

- мета роботи;
- завдання на лабораторну роботу;

– короткий виклад ходу виконання роботи з описом етапів розробки мікросервісу;

– результати роботи декількох екземплярів мікросервісу;

– графік залежності швидкодії та масштабованості від кількості використовуваних екземплярів мікросервісу та розмірності задачі.

У додаток до звіту включити:

– роздрук коду програми;

– результати роботи програми.

5.4 Контрольні питання

1) Назвіть основні архітектурні принципи сервіс-орієнтованої архітектури.

2) Назвіть основні переваги і недоліки SOA.

3) Що таке веб-сервіс?

4) Що таке мікросервіс?

5) Які принципи побудови додатків відповідно до мікросервісної архітектури?

6) Назвіть основні переваги використання мікросервісної архітектури.

7) Назвіть основні недоліки використання мікросервісної архітектури.

8) Які особливості створення мікросервісу за допомогою WSO2 Microservices Framework для Java?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1) Аксак Н.Г. Паралельні та розподілені обчислення: підручник / Н.Г. Аксак, О.Г., Руденко, А.М. Гуржій. – Х.: Компанія СМІТ, 2009. – 480с.

2) Хорстман К., Корнелл Г. Java. Библиотека профессионала, том 2. Расширенные средства, 9е изд.: пер. с англ. – М.: ООО «И.Д. Вильямс», 2014. – 1008с.

3) Таненбаум Э. С., Стин М. Распределенные системы. – М.: ДМК Пресс, 2020. – 584 с.

4) Дейтел, Х.М. Технологии программирования на Java 2: пер. с англ. Кн. 2 Распределённые приложения / Х.М. Дейтел, П.Дж. Дейтел, С.И. Сантри. – М.: Бином-Пресс, 2003. – 464 с.

5) Бэкон, Д. Операционные системы: параллельные и распределенные системы / Д. Бэкон, Т. Харрис. – СПб; К. : Питер; Изд. гр. BHV, 2004. – 799 С.

6) Бернс Б. Распределенные системы. Паттерны проектирования: пер. с англ. – СПб.: Питер, 2019. – 224 с.

7) Малявко А.А. Параллельное программирование на основе технологий OpenMP, MPI, CUDA 2-е изд., испр. и доп. Учебное пособие для вузов. – М.: Юрайт, 2019. – 116 с.

8) Ньюмен С. Создание микросервисов : пер. с англ. – СПб: Питер, 2018. – 304 с.

9) Kasun Indrasiri, Prabath Siriwardena, Microservices for the Enterprise – San Jose, CA, USA, 2018. – 434 с.

10) Christian Posta, Microservices for Java Developers – O'Reilly Media, 2015. – 129 с.

11) Boni García, Mastering Software Testing with JUnit 5 – Packt Publishing, 2017. – 399 с.