LẬP TRÌNH PHÂN TÁN DÙNG CÔNG NGHỆ MOBILE AGENT VỚI SỰ HỖ TRỢ CỦA JAVA VÀ VOYAGER

ThS. Nguyễn Khắc Quốc*

Tóm tắt

Lập trình phân tán được thực hiện bằng nhiều công nghệ khác nhau như: RMI, CORBA, DCOM, MPI, Mobile Agent, Mobile Object. Mỗi công nghệ đều có những ưu điểm và nhược điểm riêng. Công nghệ Mobile Agent với sự hỗ trợ của Java và Voyager có thể giúp chúng ta khắc phục được những hạn chế của các công nghệ trước đây trong lập trình phân tán. Việc áp dụng công nghệ này sẽ xử lý được các bài toán lớn có độ phức tạp cao mà không cần đòi hỏi vùng nhớ lớn và tốc độ xử lý nhanh. Công nghệ Mobile Agent với sự hỗ trợ của ngôn ngữ Java và Voyager vừa đơn giản vừa mang tính bảo mật cao.

Từ khóa: bảo mật cao, Java, lập trình phân tán, Mobile Agent, Voyager

Abstract

Distributed programming has been undertaken with the assistance from various technologies such as RMI, CORBA, DCOM, MPI, Mobile Agent, Mobile Object. Each has its own advantages and disadvantages. Mobile Agent with the support of Java and Voyager languages can help to overcome the drawbacks of the previous technologies in implementing distributed programming. Technology will help us to handle several complicated problems without requiring large memory and fast processing speed. Mobile Agent Technology is both simple and highly secure with the support of Java and Voyager languages.

Key words: distributed programming, highly secure, Java, Mobile Agent, Voyager

1. Giới thiệu

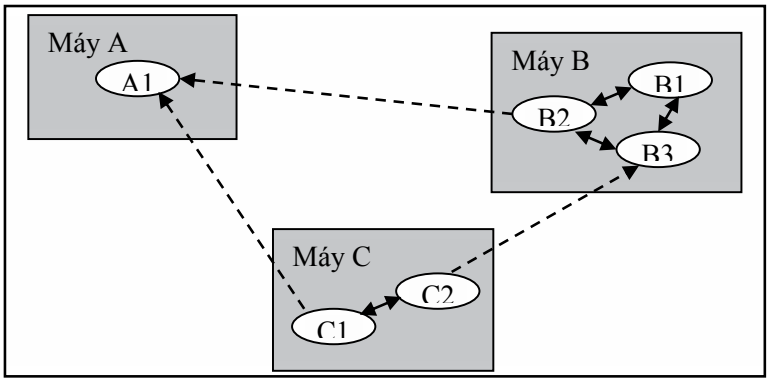
Trong thực tế, khi giải quyết một bài toán với độ phức tạp cao thì đòi hỏi không gian vùng nhớ lớn và thời gian thực hiện cần phải nhanh. Xuất phát từ những bài toán cụ thể như vậy, công nghệ lập trình phân tán đã ra đời để giải quyết vấn đề này. Hiện nay, có rất nhiều công nghệ hỗ trợ lập trình phân tán như: RMI, CORBA, DCOM, MPI, Mobile Agent, Mobile Object… Bài viết giới thiệu lập trình phân tán dùng công nghệ Mobile Agent với sự hỗ trợ của ngôn ngữ Java và Voyager.

2. Nội dung

2.1. Tìm hiểu một số công nghệ hỗ trợ lập trình phân tán

Lập trình phân tán là gì?

Thông thường các chương trình được viết dưới dạng các thủ tục hoặc hàm được nạp trực tiếp vào bộ nhớ của máy cục bộ để thực thi. Lập trình phân tán cho phép gọi một thủ tục hoặc hàm hay một đối tượng từ một máy khác trên mạng để thực hiện việc tính toán.



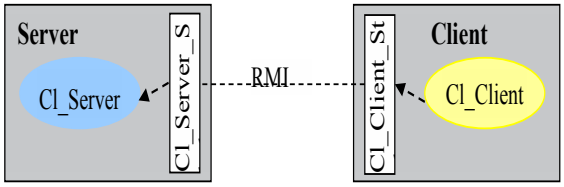
Hình 1. Minh họa mô hình lập trình phân tán

*Khoa Kỹ thuật Công nghệ - Trường Đại học Trà Vinh

2.2. Một số công nghệ hỗ trợ lập trình phân tán

2.2.1. Công nghệ RMI (Remote Method Invoke)

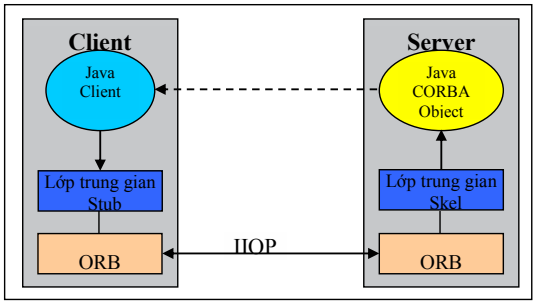
RMI là một công nghệ lập trình phân tán bằng ngôn ngữ Java.



Hình 2. Mô hình giao tiếp Client và Server

2.2.2. Công nghệ CORBA (Common Object Request Broker Architecture)

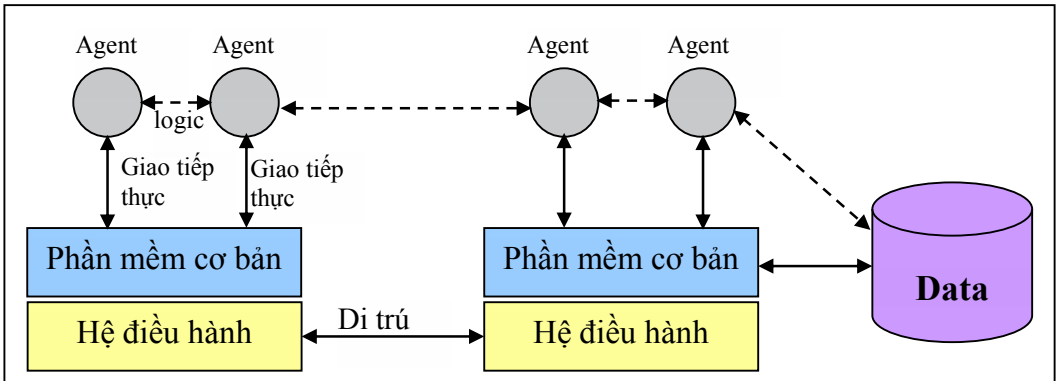
CORBA dùng một ngôn ngữ đặc tả IDL (Interface Discription Language) để đưa ra các đối tượng viết bằng những ngôn ngữ khác nhau nhưng có thể triệu gọi lẫn nhau theo mô hình phân tán.Trong CORBA, các đối tượng muốn triệu gọi được với nhau phải thông qua trình trung gian là ORB (Object Request Broker)



Hình 3. Mô hình giao tiếp hai đối tượng xa trong

2.2.3. Công nghệ Mobile Agent

Mobile Agent là một chương trình có khả năng di chuyển một cách tự trị từ một nút mạng này sang một nút mạng khác. Khi di chuyển các Mobile Agent đóng gói mã nguồn, dữ liệu và cả trạng thái thi hành, như vậy Mobile Agent có thể dừng việc đang thi hành tại máy này, di chuyển sang máy khác và khôi phục lại sự thi hành tại máy đích. Các tính chất của Mobile Agent: Tính di động; Tính tự trị; Khả năng công tác; Tính thích nghi.



Hình 4. Kiến trúc mô hình Mobile agent

Thực hiện truyền thông này theo hai cách:

- Giao tiếp thực: Hai agent giao tiếp với nhau bằng cách gửi thông điệp (các yêu cầu thủ tục) trực tiếp cho nhau.

- Giao tiếp logic: Hai agent muốn giao tiếp với nhau tạo kết nối thực với phần mềm cơ bản.

Phần mềm cơ bản của hệ thống Mobile agent gồm có 3 tầng: tầng agent, tầng an ninh và tầng truyền thông.

Tầng agent: Cung cấp tất cả các tác vụ chính cho việc thi hành và kiểm tra của tất cả các agent trên máy. Ngoài ra, nó còn cung cấp cho tất cả các agent môi trường làm việc và sự thi hành các agent độc lập với nhau.

Tầng an ninh: Cung cấp các chức năng cho phép truyền các thông điệp và các đối tượng trên mạng một cách an toàn.

Tầng truyền thông: Bao gồm các đặc tả cho các giao thức truyền, các định dạng tài liệu và các đối tượng.

Một số mô hình Mobile Agent phổ biến

Aglets: là một hệ thống Mobile agent hỗ trợ các khái niệm thi hành tự trị và định tuyến động trên lộ trình của nó. Aglets server là chương trình cung cấp môi trường thi hành và một máy ảo Java cho aglets hoạt động. Aglets sử dụng giao thức ATP (Aglets Transfer Protocol) cho việc di chuyển và giao tiếp.

Voyager: Là một môi trường thương mại hỗ trợ phát triển các ứng dụng agent được Object Space phát triển.

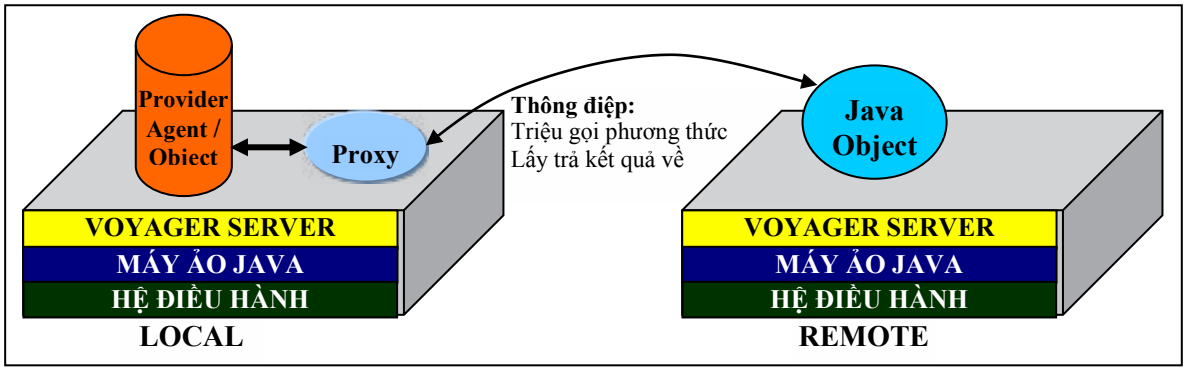
Mole: Là hệ thống Mobile agent được xây dựng với ngôn ngữ Java.

Telescript: Là một sản phẩm thương mại được phát triển bởi General Magic Incorporated để hỗ trợ cho Aobile agent trong môi trường thương mại điện tử.

2.3. Voyager trong lập trình phân tán

Voyager là một công nghệ hỗ trợ lập trình phân tán (Mobile Object và Mobile Agent) sử dụng ngôn ngữ lập trình Java với cú pháp chuẩn hỗ trợ lập trình tạo dựng các đối tượng từ xa một cách dễ dàng, cho phép các ứng dụng này trao đổi thông điệp với nhau và di chuyển các đối tượng giữa các máy tính có hỗ trợ Voyager.

Voyager hỗ trợ di động mạnh có khả năng mang toàn bộ mã chương trình và dữ liệu di chuyển từ máy ảo Java (JVM) này sang máy ảo Java khác nếu các máy có hỗ trợ Voyager. Trạng thái của các agent cũng được bảo toàn và tiếp tục thực thi tại nơi agent đến. Các chương trình viết trong Voyager có thể giao tiếp với các chương trình viết bằng SOAP, CORBA, RMI và DCOM. Điểm mạnh của Voyager là đơn giản, dễ dùng, che giấu các kỹ thuật lập trình phân tán phức tạp.



Hình 5. Mô hình hoạt động của Voyager

Để các agent hoạt động thì cần phải có một môi trường hỗ trợ Voyager Server ở cả hai phía (máy cục bộ và máy ở xa)

Khi một đối tượng từ xa được tạo ra ở máy cục bộ (nhờ vào Provider Agent/Object) thì một Proxy sẽ tự động được sinh ra. Voyager Server trên máy cục bộ sẽ giao tiếp với đối tượng từ xa thông qua Proxy. Giao tiếp là việc trao đổi thông điệp giữa đối tượng từ xa và Server cục bộ.

2.4. Xây dựng ứng dụng phân tán Voyager 1.0.1

Xây dựng một ứng dụng phân tán ta cần trải qua hai bước sau:

Bước 1: Cài đặt lớp xử lý. Lớp này sau khi biên dịch sẽ là lớp dùng để sinh ra các đối tượng từ xa.

Bước 2: Cài đặt lớp giao tiếp người sử dụng.

Ví dụ: Để tạo ứng dụng phân tán dùng tính tổng các số từ 1 đến N thực hiện các bước sau:

Bước 1: Tạo lớp giả sử là CountMobile. Lớp này chứa phương thức public long test (long a, long b): dùng để tính tổng từ a đến b. Dùng trình javac để biên dịch lớp này thành CountMobile.class. Sau đó, dùng trình vcc để biên dịch CountMobile.class, lúc này sẽ sinh ra file VCountMobile.java. Lớp này do voyager tự sinh ra.

Bước 2: Cài đặt lớp CountMobileClient để sử dụng lớp VCountMobile. Lớp này sẽ giao tiếp với người dùng, bao gồm việc tạo đối tượng từ xa, gửi đối tượng đi, yêu cầu xử lý, nhận kết quả trả về và hủy đối tượng từ xa.

*** Code minh họa:**

Tạo lớp tính tổng từ a đến b

```
public class CountMobile implements java.
io.Serializable {
    String name;
    public CountMobile() {
    }
    public long test(long a, long b) {
        long kq=0;
        for (long i=a ; i<=b ; i++) {
```

```
            kq += i;
        }
System.out.println("Da tinh xong tong tu: " + a + " -> " + b + " , KQ: " + kq);
        return kq;
}
}
```

Dùng javac để biên dịch lớp trên ta được file: CountMobile.class

Dùng vcc để biên dịch ta được tập tin: VCountMobile.java

Tạo lớp giao tiếp với người dùng

```
import java.io.*;
import java.io.BufferedInputStream;
import COM.objectspace.voyager.*;
public class CountMobileClient{
public static void main(String args[]) {
String add[];
long SUM = 13;
BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
        int num, i, numObj=1, succ[], realObj;
        long kq=0;
        try{
        System.out.println("\n---------- TINH TONG TU 1 -> N ----------");
                System.out.print("Nhap so N can tinh: ");
                SUM = Integer.parseInt(bf.readLine());
        System.out.print("Ban muon chia cong viec cho bao nhieu may?");
                numObj = Integer.parseInt(bf.readLine());
                } catch(Exception e){}
        succ = new int[numObj];
        add = new String[numObj];
        for(i=0; i<numObj; succ[i]=1, i++ );
try {
                VCountMobile myMCount[];
        myMCount = new VCountMobile[numObj];
        for(i=0; i<numObj; i++){
                myMCount[i] = new VCountMobile("localhost");
        }
        realObj = numObj;
```

```
        for(i=0; i<numObj; i++){
        System.out.print("Nhap dia chi (IP va Port) cua may thu " + (i+1) +": ");
try{
                        add[i] = bf.readLine();
                } catch(Exception e){}
                try{
                        myMCount[i].moveTo(add[i]);
                }
                catch(VoyagerException e){
                        succ[i] = 0;
                        realObj--;
                        System.out.println("Khong tim thay server: " + add[i]);
                }
        }
        long khoang = SUM / realObj, so=0;
        long kt =1;
        int vt=0;
System.out.println("\n---- KET QUA TINH TOAN TU CAC MAY -------\n");
        for(i=0; i<numObj; i++){
                if(succ[i]==1){
                        kt = so+1;
                        so = so + khoang;
                        try{
                                long temp = myMCount[i].test(kt, so);
System.out.println("Ket qua lay tu may: " + add[i] + " la: " + temp);
                                kq +=  temp;
                                vt = i;
                        }
                        catch(TransportException e){
System.out.println("Khong the lay Ket qua tu may: " + add[i]);
                        }
                }
        }
        if(SUM % realObj != 0){
                long temp = myMCount[vt].test(so+1, SUM);
System.out.println("Ket qua lay tu may: " + add[vt] + " la: " + temp);
```

```
                kq +=  temp;
        }
    System.out.print("Ket qua sau khi tinh toan: ");
    System.out.println(kq);
          Voyager.shutdown();
      } catch (VoyagerException exception) {
        System.err.println(exception);
      }
    }
}
```

Biên dịch lớp CountMobileClient và VcountMobile. Chạy máy ảo Java để kiểm tra chương trình.

*** Cài đặt voyager 1.0.1**

1. Cài đặt bộ JDK 1.3

2. Chạy Setup Voyager1.0.1.exe. Cài vào C:\voyager1.0.1

3. Thêm biến môi trường CLASSPATH = :\voyager1.0.1\lib\voyager1.0.1.jar.

4. Dịch chương trình

- Dịch: javac <tên file>.java.
- Dịch vcc: <tên file>.class
- Dịch tất cả: javac *.java

5. Chạy chương trình

- Chạy Server: voyager 9000 (Chạy trên cổng 9000).
- Chạy client java <tên file>

3. Kết luận

Như vậy, khi dùng Mobile Agent với sự hỗ trợ của Java và Voyager ta thấy rằng có nhiều ưu điểm hơn so với dùng công nghệ RMI, CORBA, DCOM…

Mobile Agent có khả năng di chuyển một cách tự trị từ một nút mạng này sang một nút mạng khác; đóng gói mã nguồn, dữ liệu và cả trạng thái thi hành.Việc lập trình đơn giản, chỉ cài đặt ứng dụng ở một phía cục bộ. Đặc biệt, Voyager còn có khả năng mang toàn bộ mã chương trình và dữ liệu di chuyển từ máy ảo Java này sang máy ảo Java khác nếu các máy có hỗ trợ Voyager mà các công nghệ khác không hỗ trợ được.

Tài liệu tham khảo

Nguyễn Lê Anh. 2005. *Giáo trình xử lý song song,* Học viện Kỹ thuật Quân sự.

Đoàn Văn Ban. 2005. *Giáo trình Lập trình Java.* Đại học Sư phạm Hà Nội.

David Reilly- Michael Reilly. 2002. *Java Network Programming and Distributed Computing.* Publisher Addison Wesley. Pub Date March 25. ISBN: 0-201-71037-4

John Wiley & Sons. 2004. *Tools and Environments for parallel and Distributed computing.* All rights reserved. Published by John Wiley & Sons, Inc, Hoboken, New Jersey. Published simultaneously in Canada.

GeorgeCoulouris et al. 2005. *Distributed systems: concept and design.*