Documentazione Tecnica del Progetto: Piattaforma di Clustering Gerarchico Distribuito

Lascaro Gianluca
Matricola: 758367

Esame di Metodi Avanzati di Programmazione - Anno Accademico 2023/24
Aprile 2025

Indice

Introduzione

Il progetto realizza un sistema distribuito **client-server** per l'analisi avanzata di dataset tramite algoritmi di **clustering gerarchico**. L'applicazione consente di:

- Caricare dati da database MySQL
- Generare dendrogrammi con algoritmi **Single-Link** e **Average-Link**
- Visualizzare, salvare e ricaricare modelli serializzati
- Gestire connessioni concorrenti tramite architettura multithreading

Il sistema integra componenti di **elaborazione distribuita**, **GUI interattiva** (Java Swing), e **persistenza dati**, seguendo i principi del pattern **MVC** (Model-View-Controller).

1 Scopo e Obiettivi

1.1 Obiettivi Funzionali

- **Comunicazione Client-Server**:
  - Connessione TCP/IP con gestione di porte customizzabili
  - Serializzazione di oggetti complessi (dendrogrammi, dataset) tramite `ObjectOutputStream`
- **Clustering Gerarchico**:
  - Supporto per **distanze Single-Link** (minima tra cluster) e **Average-Link** (media tra cluster)
  - Costruzione di dendrogrammi con profondità personalizzabile
- **Integrazione Database**:
  - Lettura di tabelle MySQL e conversione in oggetti `Data` per l'elaborazione
- **Gestione File**:
  - Salvataggio e caricamento di dendrogrammi in formato binario (`.ser`)

1.2 Obiettivi Non Funzionali

- **Scalabilità**: Server multithreading per gestire più client
- **Robustezza**: Gestione centralizzata di eccezioni (es. `InvalidDepthException`, `NoDataException`)
- **Usabilità**: Interfaccia grafica intuitiva con validazione input

2 Architettura del Sistema

2.1 Diagramma dei Componenti

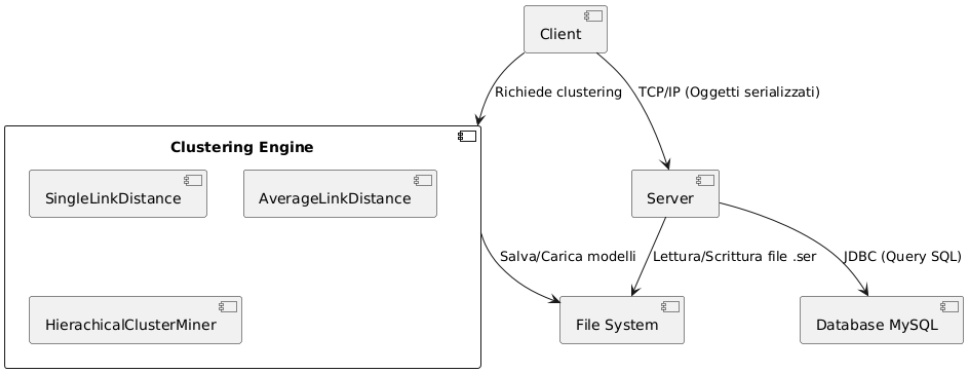

Figura 1: Diagramma dei componenti principali e loro interazioni

2.2 Diagramma delle Classi

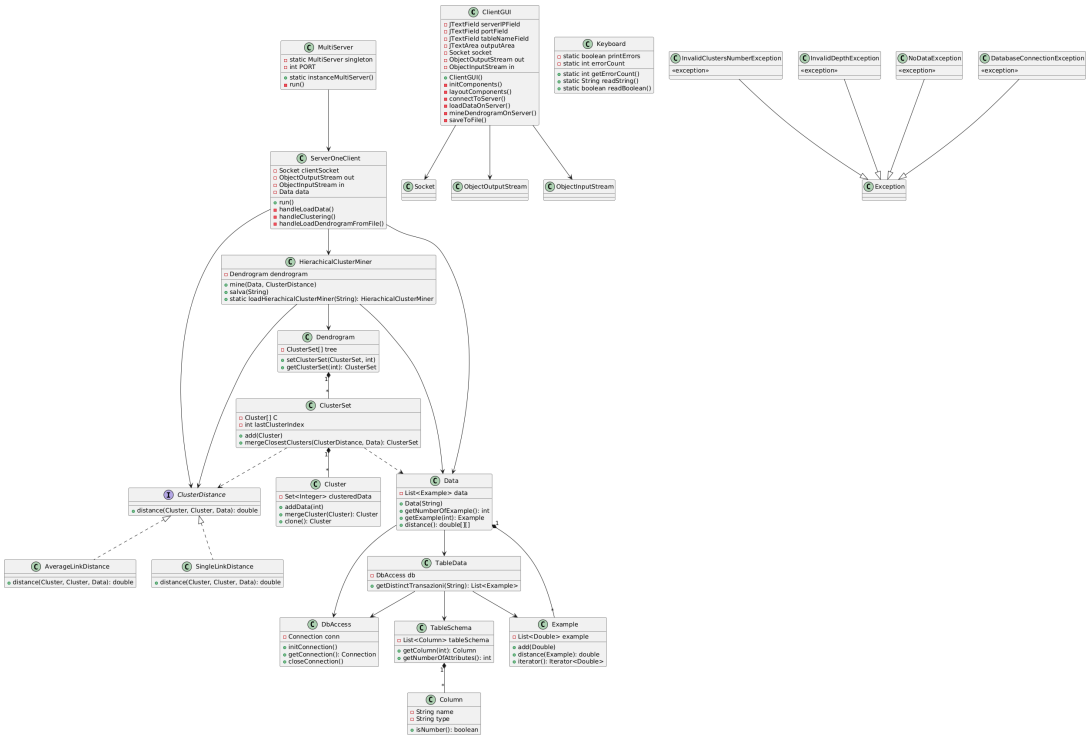

Figura 2: Struttura dettagliata delle classi e delle loro relazioni

Principali componenti:

- **ClientGUI**: Gestione interfaccia utente e comunicazione
- **MultiServer**: Implementazione pattern Singleton per il server
- **HierarchicalClusterMiner**: Core algorithm implementation
- **ClusterDistance**: Gerarchia per le strategie di calcolo distanze

3 Flusso di Lavoro (Workflow)

3.1 Diagramma di Sequenza

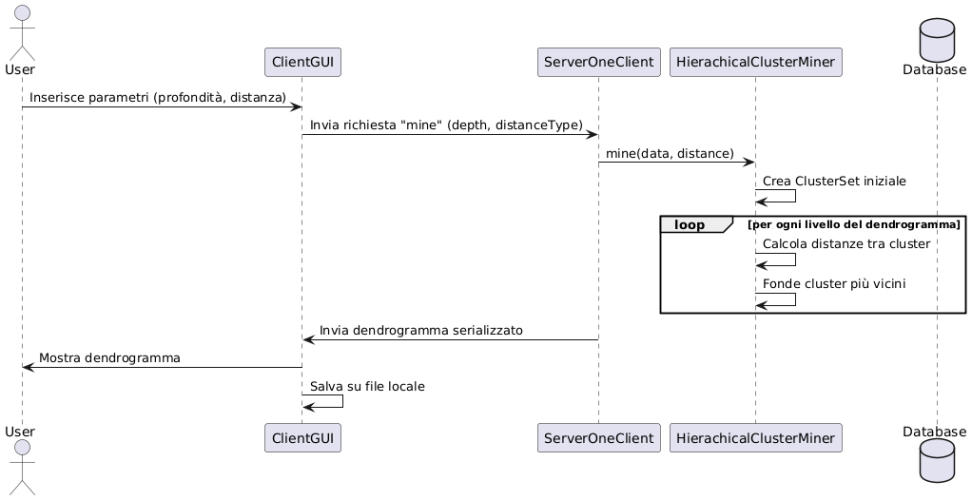

Figura 3: Interazione client-server durante il processo di clustering

3.2 Diagramma di Attività

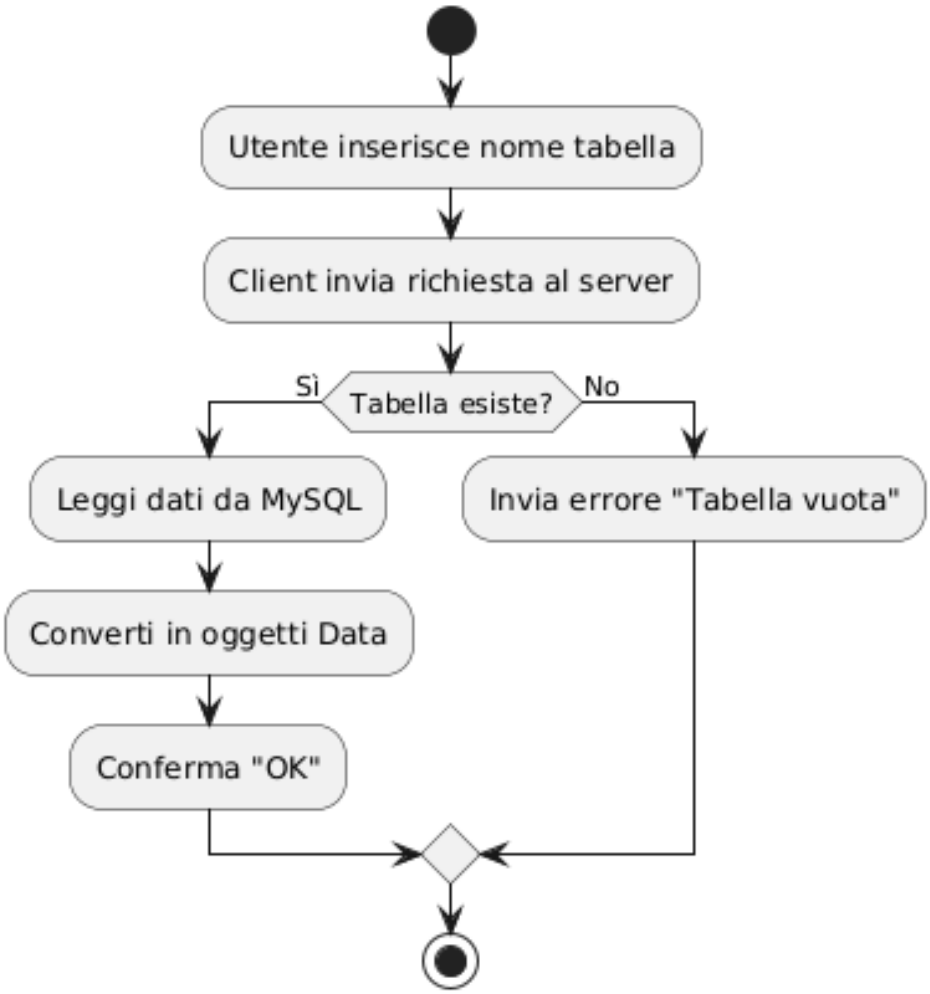

Figura 4: Flusso decisionale per la generazione del dendrogramma

4 Dettagli Implementativi

4.1 Struttura del Client

Il client implementa un'interfaccia grafica Swing basata sul pattern MVC semplificato:

- **View**: Componenti Swing (JFrame, JButton, JTextArea)
- **Controller**: Gestori eventi e logica di comunicazione

- **Model**: Dati ricevuti dal server (dendrogrammi)

4.1.1 Architettura GUI

```
public class ClientGUI extends JFrame {
    // Componenti UI
    private JTextField serverIPField, portField, tableNameField;
    private JTextArea outputArea;
    private JButton connectButton, loadDataButton;

    // Gestione connessione
    private ObjectOutputStream out;
    private ObjectInputStream in;
    private Socket socket;

    // Stato applicazione
    private boolean isConnected = false;
    private boolean isDataLoaded = false;
    private String dendrogramToSave = null;
}
```

Listing 1: Struttura principale del ClientGUI

4.1.2 Organizzazione dei Componenti

La GUI è strutturata in tre sezioni principali:

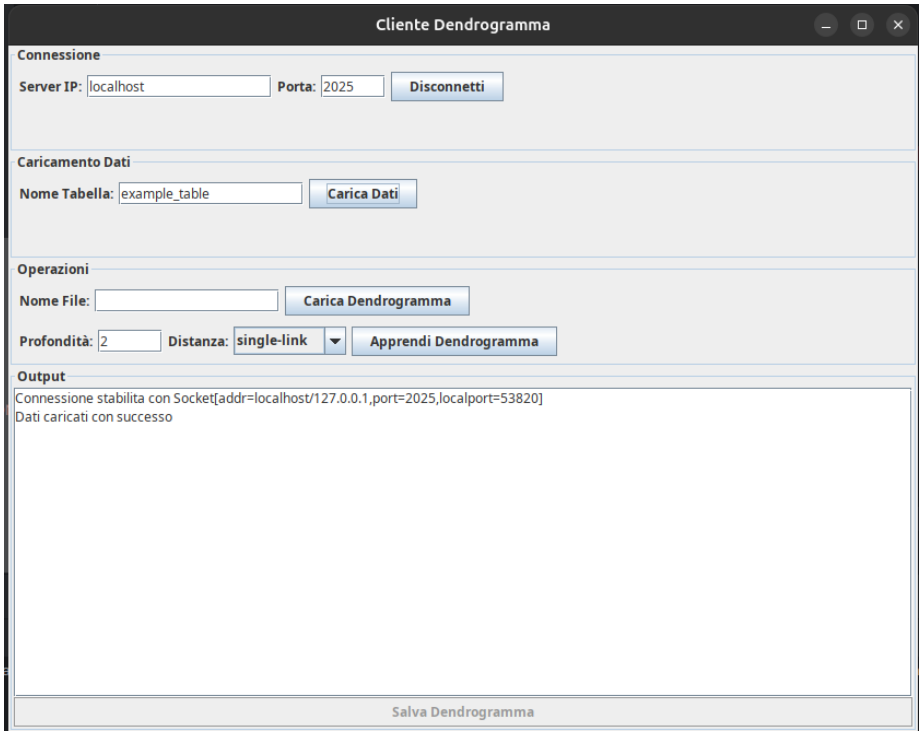

Figura 5: Layout dell'interfaccia client con aree funzionali

- **Connessione**: Configurazione IP/Porta server
- **Caricamento Dati**: Selezione tabelle da database MySQL
- **Operazioni**: Scelta algoritmi e parametri di clustering
- **Output**: Visualizzazione dendrogrammi e salvataggio

4.1.3 Gestione Eventi e Comunicazione

Flusso di interazione tipico:

1. Connessione al server

```
private void connectToServer() {
    try {
        socket = new Socket(serverIPField.getText(),
            Integer.parseInt(portField.getText()));
        out = new ObjectOutputStream(socket.getOutputStream());
        in = new ObjectInputStream(socket.getInputStream());
        // Abilita funzionalita post-connessione
    } catch (IOException ex) {
        outputArea.append("Errore di connessione: " + ex.getMessage());
```

```
    }
}
```

2. Caricamento dati da tabella MySQL

```
private void loadDataOnServer() {
    out.writeObject(0); // Codice operazione
    out.writeObject(tableNameField.getText().trim());
    String response = (String) in.readObject();
    if(response.equals("OK")) isDataLoaded = true;
}
```

3. Esecuzione clustering

```
private void mineDendrogramOnServer() {
    int depth = Integer.parseInt(depthField.getText());
    int distanceType = distanceTypeComboBox.getSelectedIndex() + 1;

    out.writeObject(1); // Codice operazione
    out.writeObject(depth);
    out.writeObject(distanceType);

    dendrogramToSave = (String) in.readObject();
    outputArea.setText(dendrogramToSave);
}
```

4.1.4 Caratteristiche Principali

- **Gestione Connessione Persistente**:
  - Singleton Socket/Streams
  - Riconnessione automatica
  - Chiusura graceful su WindowEvent
- **Validazione Input**:
  - Controllo formati numerici (porte, profondità)
  - Check campi obbligatori
  - Gestione errori tramite JOptionPane
- **Serializzazione**:
  - Scambio oggetti tramite ObjectOutputStream
  - Salvataggio locale dendrogrammi
  - Compatibilità con formato .ser

4.1.5 Gestione Errori Avanzata

- Eccezioni custom per stati invalid:

```
if (!isConnected || !isDataLoaded) {
    outputArea.append("Operazione non disponibile: ");
    outputArea.append("- Connessione: " + isConnected);
    outputArea.append("- Dati caricati: " + isDataLoaded);
}
```

Gestione Connessione:

```
private void connectToServer() {
    if (isConnected) closeConnection();
    try {
        socket = new Socket(serverIPField.getText(), Integer.parseInt(
            portField.getText()));
        out = new ObjectOutputStream(socket.getOutputStream());
        in = new ObjectInputStream(socket.getInputStream());
        // Abilita pulsanti post-connessione
    } catch (IOException e) {
        outputArea.append("Errore: " + e.getMessage());
    }
}
```

4.2 Architettura del Server

La componente server dell'applicazione è progettata per gestire connessioni multiple e per eseguire algoritmi di clustering gerarchico in modo parallelo. Il server sfrutta un'architettura multithread e un pattern Singleton per garantire scalabilità ed efficienza nell'elaborazione delle richieste client.

Componenti chiave

- `MultiServer`: Gestione del pool di connessioni in ingresso. Implementa il pattern Singleton per garantire un'unica istanza del server.
- `ServerOneClient`: Estende `Thread` e gestisce ciascun client in modo indipendente. Riceve ed elabora richieste come caricamento dati, clustering, operazioni su file.
- `ThreadPool`: Opzionale, può essere utilizzato per una gestione ottimizzata delle risorse e del numero di thread attivi.

Gestione Connessioni e Richieste

Il server principale accetta connessioni da nuovi client e crea un nuovo thread per ognuno:

```
public class MultiServer {
    private static MultiServer instance;
    private MultiServer(int port) { /* ... */ }

```

```
    public static void instanceMultiServer() {
        if (instance == null)
            instance = new MultiServer(2025);
        instance.run();
    }

    private void run() {
        while (true) {
            Socket socket = serverSocket.accept();
            new ServerOneClient(socket).start(); // Nuovo thread per client
        }
    }
}
```

Ogni thread client esegue in ciclo continuo la lettura delle richieste:

```
class ServerOneClient extends Thread {
    public void run() {
        while(true) {
            int requestType = in.readObject();
            switch(requestType) {
                case 0: handleLoadData(); break;
                case 1: handleClustering(); break;
                case 2: handleFileOperation(); break;
                default: /* gestione errori */ break;
            }
        }
    }
}
```

5 Strutture Dati e Algoritmi di Clustering

5.1 Gerarchia dei Cluster

Il sistema utilizza tre componenti fondamentali per rappresentare la struttura gerarchica:

- **Cluster**: Unità base che rappresenta un gruppo di elementi
- **ClusterSet**: Collezione di cluster a uno specifico livello gerarchico
- **Dendrogramma**: Struttura ad albero che memorizza l'evoluzione dei cluster

5.1.1 Classe Cluster

Rappresenta un singolo cluster con operazioni di base:

```
public class Cluster implements Iterable<Integer>, Cloneable, Serializable
    {
    private Set<Integer> clusteredData = new TreeSet<>();

```

```
    // Fusione di due cluster
    Cluster mergeCluster(Cluster c) {
        Cluster newC = new Cluster();
        Iterator<Integer> it1 = this.iterator();
        Iterator<Integer> it2 = c.iterator();
        while(it1.hasNext()) newC.addData(it1.next());
        while(it2.hasNext()) newC.addData(it2.next());
        return newC;
    }

    // Altre operazioni: addData, getSize, clone, etc.
}
```

Listing 2: Implementazione della classe Cluster

5.1.2 Classe ClusterSet

Gestisce gruppi di cluster a un determinato livello:

```
ClusterSet mergeClosestClusters(ClusterDistance distance, Data data) {
    // Calcola distanze tra tutte le coppie
    for(int i=0; i<this.C.length; i++) {
        for(int j=i+1; j<this.C.length; j++) {
            double d = distance.distance(get(i), get(j), data);
            // Trova la coppia con distanza minima
        }
    }
    // Crea nuovo ClusterSet con cluster fusi
}
```

Listing 3: Metodo per la fusione dei cluster

5.2 Metriche di Distanza

Il sistema implementa due strategie per il calcolo delle distanze tra cluster:

- **Single-Link**:
$$d(C_1, C_2) = \min_{a \in C_1, b \in C_2} \text{distanza}(a, b)$$
- **Average-Link**:
$$d(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{a \in C_1} \sum_{b \in C_2} \text{distanza}(a, b)$$

5.3 Processo di Clustering Gerarchico

L'algoritmo agglomerativo segue questi passi:

1. **Inizializzazione**: Creazione di cluster singleton

```
    ClusterSet level0 = new ClusterSet(data.size());
    for(int i=0; i<data.size(); i++) {
        Cluster c = new Cluster();
        c.addData(i);
        level0.add(c);
    }
```

2. **Calcolo Matrice Distanze**

3. **Fusione Iterativa**: Unione progressiva dei cluster più vicini

```
    for(int level=1; level<depth; level++) {
        ClusterSet newLevel = previousLevel.mergeClosestClusters(
            distance, data);
        dendrogram.setClusterSet(newLevel, level);
    }
```

4. **Aggiornamento Gerarchia**: Costruzione del dendrogramma

5.4 Classe HierarchicalClusterMiner

Coordina il processo di clustering implementando il pattern Strategy:

```
public void mine(Data data, ClusterDistance distance) {
    // Inizializza il livello 0
    ClusterSet initial = createInitialClusters(data);

    // Processo iterativo
    for(int level=1; level<depth; level++) {
        ClusterSet merged = mergeClosestClusters(
            dendrogram.getClusterSet(level-1),
            distance,
            data
        );
        dendrogram.setClusterSet(merged, level);
    }
}
```

Listing 4: Metodo principale di mining

5.5 Descrizione delle Schermate

L'interfaccia grafica del client è organizzata in tre schermate principali che guidano l'utente attraverso il workflow di clustering. Di seguito una descrizione dettagliata.

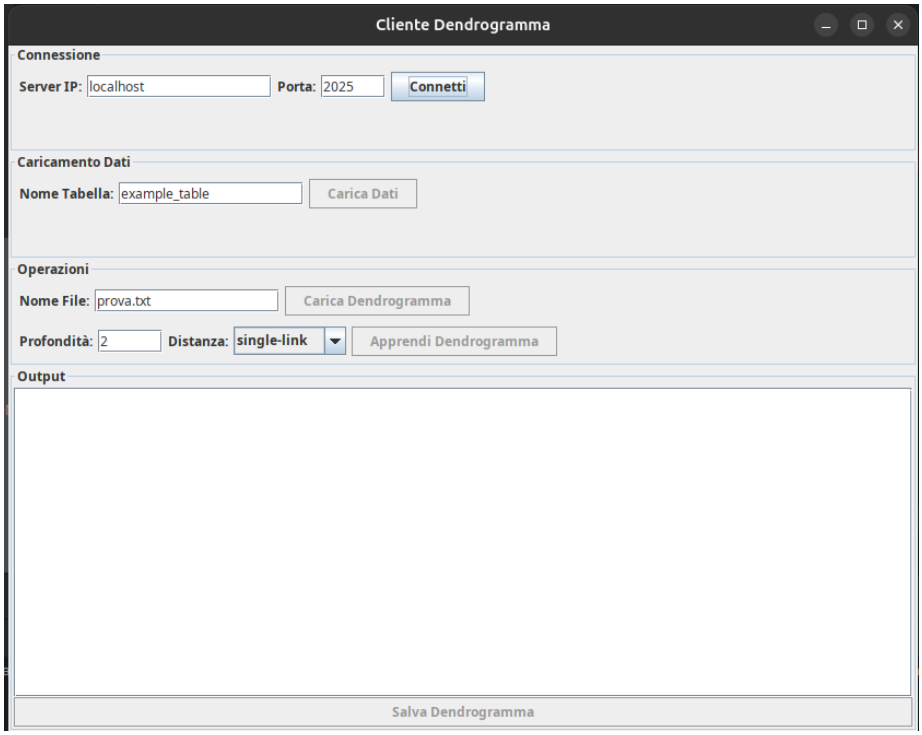

Figura 6: Schermata iniziale di connessione e configurazione. L'utente può specificare l'IP del server (default: localhost), la porta (2025), e avviare la connessione. Sono presenti sezioni per il caricamento di dati da tabelle MySQL e l'impostazione dei parametri di clustering.

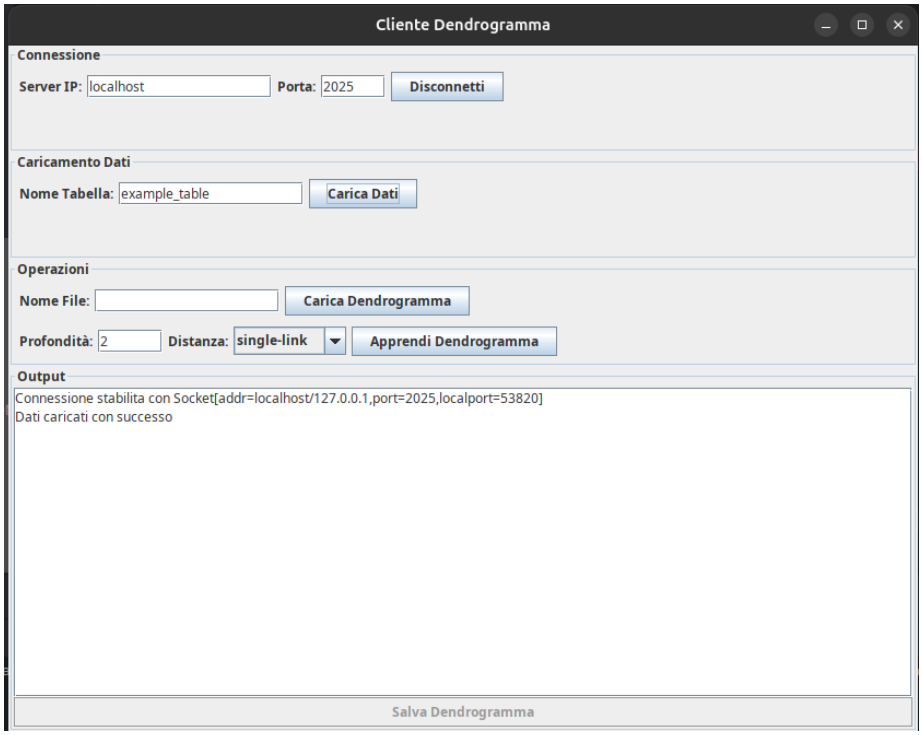

Figura 7: Schermata operativa post-connessione. Mostra la conferma della connessione stabilita e l'esito del caricamento dati. L'utente può procedere alla generazione del dendrogramma impostando profondità (es. 2) e metrica di distanza (single-link).

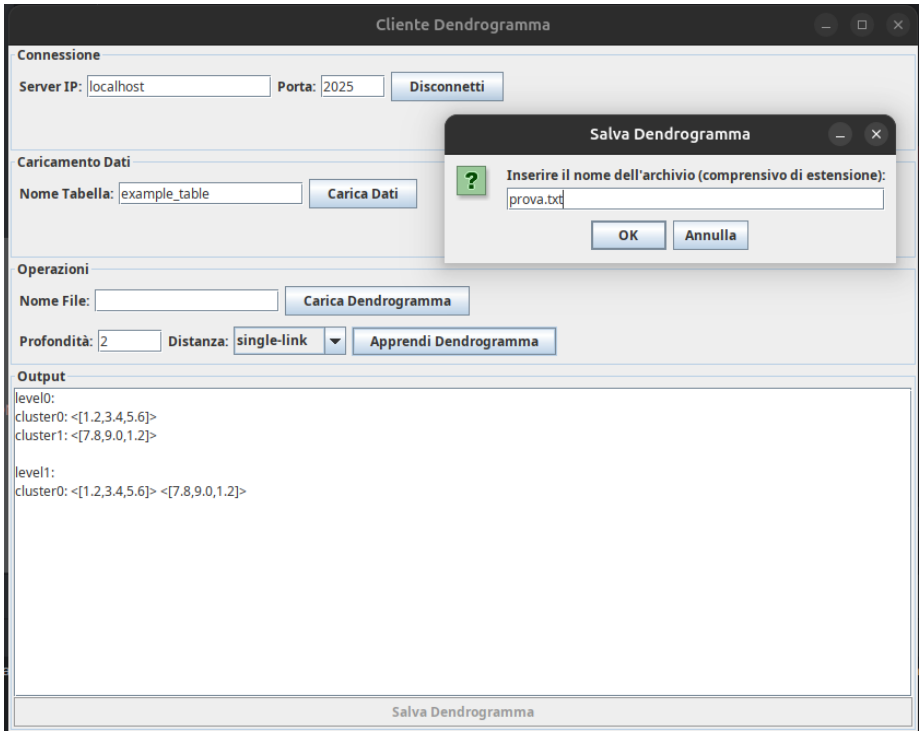

Figura 8: Risultato del clustering gerarchico. Visualizza i cluster a diversi livelli di profondità (level0 e level1), con i relativi centroidi. Include opzioni per salvare il dendrogramma in formato binario o annullare l'operazione.

6 Gestione degli Errori

6.1 Eccezioni Personalizzate

- `InvalidDepthException`: Profondità non valida (es. 0)
- `NoDataException`: Tabella database vuota o inesistente
- `InvalidClustersNumberException`: Tentativo di fusione con un solo cluster

6.2 Logging e Feedback

- Output in `JTextArea` con messaggi di errore dettagliati
- Utilizzo di `JOptionPane` per dialoghi modali in caso di errori critici
- Sistema di logging centralizzato nel server

7 Deployment e Configurazione

7.1 Requisiti di Sistema

- **MySQL Server**: Configurazione con tabelle secondo schema specificato
- **Librerie**: MySQL Connector/J inclusa nel classpath

7.2 Configurazione Database

- Creare un utente `root` con privilegi sulla tabella target
- Esempio di struttura tabella:

```
CREATE TABLE example_table (
    feature1 FLOAT,
    feature2 FLOAT,
    feature3 FLOAT
);
```

Conclusioni

Il progetto dimostra un'implementazione robusta di un sistema distribuito per l'analisi dati, combinando tecnologie avanzate (socket, multithreading, algoritmi di clustering) in un'architettura modulare. Le scelte progettuali, come l'uso della serializzazione per i dendrogrammi e il pattern Singleton per il server, garantiscono scalabilità e manutenibilità.