Rapport d’évaluation du service Hdfs

Martin Dubois / Yannis Mehidi

Rapport d’évaluation du service Hdfs (Martin/Yannis)

Les tests

Présentation des tests

Tests unitaires pour:

- DataNode
- InfoFichier
- NameNode

Test de communication entre HdfsServer(du NameNode) et HdfsClient

Test de communication entre le HdfsClient et le HdfsServer(des DataNodes sur d'autres machines).

Test global de hdfs.

Resultat des tests

DataNode

(Voir Annexe DataNodeTests)

La création de fichiers ne met pas les fichiers dans le Path que l’on à défini en créant le DataNode donc il n’arrive pas à les retrouvés. Ceci peut être résolu très simplement transformant

```
Path file = Paths.get(this.makeName(fname, chunk));
```

en `Path file = Paths.get(this.dir.getPath() + "\\" + this.makeName(fname, chunk));`

dans la fonction addChunk (ligne 90).

Si on effectue cette petite modification tous les tests fonctionnent.

InfoFichier

(Voir Annexe InfoFichierTests)

Pas de problèmes, tous les tests fonctionnent.

Rapport d’évaluation du service Hdfs (Martin/Yannis)

NameNode

(voir Annexe NameNodeTests)

Le getInfoFichier ne fonctionne pas car il vérifie l'égalité entre 2 strings avec == au lieu d'utiliser equals() . Ceci peut être résolu très simplement transformant

```
if (f.getNom()==nom)
```

en `if (f.getNom().equals(nom))` Dans la fonction GetInfoFichier (ligne 40).

Si on effectue cette petite modification tous les tests fonctionnent.

Communication entre HdfsServer (NameNode) et HdfsClient

(Voir figure 2)

On remarque que le client et le NameNode communiquent dans les deux sens sans erreurs.

Le NameNode gère même les erreurs (dans le cas où le fichier n'est pas présent il le signale en envoyant un message d'erreur).

Communication entre HdfsServer (DataNodes) et HdfsClient

(Voir figure 3)

On remarque que le client peut communiquer avec les dataNodes (si on ajoute une fonction créant une requête du HdfsClient vers les DataNodes).

Test de Hdfs

Le test ne peut pas être fait, les méthodes Write et Delete de HdfsClient ne sont pas implantés, de plus la méthode Read n’est pas complète.

=> Le back-end et les systèmes de communications fonctionnent donc, seule l’implémentation de HdfsClient n’est pas faite.

Rapport d’évaluation du service Hdfs (Martin/Yannis)

Synthèse

Correction

Le produit ne fonctionne pas car l'interfaces front-end (hdfsClient) n'est pas implémenté.

Cependant le reste fonctionne dans sa globalité, les tests précédents montrent que le back-end (DataNode / NameNode et autres classes utilitaires) fonctionne outre quelques bugs mineurs.

Complétude

Ce qu'il manque complètement :

- HdfsClient, les méthodes HdfsDelete/HdfsWrite ne sont pas implémentés.
- DataNode, il manque un Slave pour assurer le keepAlive.

Ce qui n'est pas complet :

- HdfsRead pour le HdfsClient, il manque la requête des chunks aux DataNodes (il ne recupère que la liste des chunks et leurs adresses) .
- GetChunk pour le HdfsServer (qui sert donc à récuperer un chunk du DataNode pour l'envoyer au client).
- HdfsQuery, pas de querry pour delete.

Pertinence

- Les dataNodes devraient s'inscrire auprès de NameNode (on ne devrais pas avoir à déclarer manuellement les adresses des DataNodes)
- Il devrait être possible d'avoir un NameNode et un HdfsClient sur 2 machines différentes.

- On a un bien un système de client – serveur comme demandé.
- La transmission d'information passe par des sockets comme demandé.

Cohérence

Bien que tout ne soit pas implémenté il y a tout de même les méthodes qui sont déclarées.

Pas de redondance, l'architecture est relativement simple et efficace (voir Figure.1 dans l'annexe).

Les méthodes disponibles pour DataNode et NameNode sont completes, sans redondance.

Rapport d’évaluation du service Hdfs (Martin/Yannis)

Les méthodes de HdfsServer et HdfsClient sont correctes et correspondent aux attentes du sujet (pouvoir Read/Write et Delete un fichier sur le service hdfs), cependant il n'y a pas de message HdfsQuery pour requêter un delete.

Séparation claire entre le front-end et le back-end : entre l'interface de communication réseau (HdfsServer.java) et le back-end (DataNode.java / NameNode.java).

NameNode possède un SlaveKeepAlive qui vérifie toutes les 5 secondes que les DataNodes sont toujours vivants. Par contre DataNode ne possède pas de Slave envoyant des messages de KeepAlive au NameNode.

=> L'architecture est simple, claire, efficace. Elle respecte les demandes du sujet et les bonnes pratiques de séparation des tâches entre le front-end et le back-end.

Pistes d'amélioration

Dans le NameNode on pourrait changer le booléen de liste_fichiers_occupes en une énumération pour permettre de différencier l'occupation par écriture et lecture afin de faire de la lecture en parallèle.

Remplacer les messages d'erreurs par des exceptions afin de rendre le code plus clair et le traitement des erreurs plus simple, notamment dans le HdfsServer.

Le -h / --help de HdfsServer qui explique les arguments de lancement de hdfsServer n'est pas clair du tout, voici un meilleur message d'explication :

-h / --help → Explique comment utiliser les arguments du programme

-n / namenode / nameNode → Permet de lancer un NameNode

-f + Path_fichier_conf → Permet de lancer un NameNode à partir d'un fichier de configuration

sans arguments → Permet de lancer un DataNode

Remplacer les parcours sur les listes (dans NameNode) par des liste.remove() et liste.contains().

Il faudrait gérer l'accès concurrent à la liste des dataNodes (par le SlaveKeepAlive et les HdfsClient) du NameNode.

Annexe

InfoFichierTests

```
public class InfoFichierTests {

	@Test
	public void test() throws UnknownHostException {
		InfoFichier c = new InfoFichier("test");
		Hashtable<Integer,Inet4Address> a = new Hashtable<Integer,Inet4Address>();
		a.put(0,(Inet4Address)Inet4Address.getLocalHost());
		a.put(3,(Inet4Address)Inet4Address.getLoopbackAddress());

		// Test de AddChunk
		c.addChunk(0, (Inet4Address)Inet4Address.getLocalHost());
		c.addChunk(3, (Inet4Address)Inet4Address.getLoopbackAddress());
		assertEquals(c.getNom(),"test");

		// Test de GetChunks
		assertEquals(c.getChunks(),a);
	}



}
```

DataNodeTests

Attention ces tests on été écrits pour Windows (pour linux il faut changer le PATH et remplacer les « \\ » par des « / »).

```
public class DataNodeTests {

	private String file = "C:\\Users\\Martin\\Documents\\ENSEEIHT\\Hdp\\hdp-master\\
data";

	@Test(expected=NotDirectoryException.class)
	public void testDataNodeErreurExistePas() throws NotDirectoryException {
		DataNode data = new DataNode("");
	}
	@Test(expected=NotDirectoryException.class)
	public void testDataNodeErreurPasRepertoire() throws NotDirectoryException {
		DataNode data = new DataNode(file+"\\filesample.txt.1");
	}
	@Test
	public void testDataNode() throws NotDirectoryException {
		DataNode data = new DataNode(file);
	}
	@Test
	public void testAddGetDelChunk() throws IOException {
		DataNode data = new DataNode(file);
		String s = "Ceci est un test nécessaire\nEt il doit fonctionner!";

		// Test Add
		data.addChunk("test.txt", 1, s);
		data.addChunk("test.txt", 2, s);
		File f = new File(file+"\\test.txt.1");
		File ff = new File(file+"\\test.txt.2");
		if (!f.exists()&&!ff.exists()) {
			fail();
		}

		// Test Get
		String ss = data.getChunk("test.txt", 1);
		System.out.print(ss);
		System.out.println("");
		System.out.print(s);

		// Test Del
		data.delChunk("test.txt", 1);
		data.delChunk("test.txt", 2);
		if (f.exists()&&ff.exists()) {
			fail();
		}

	}

}
```

NameNodeTests

```
public class NameNodeTests {

	@Test
	public void testAjouterFichier_GetInfoFichier() throws UnknownHostException,
AlreadyExists {
		NameNode nm = new NameNode();
		InfoFichier f = new InfoFichier("f.txt");
		f.addChunk(1, (Inet4Address)Inet4Address.getLocalHost());

		// Test de AjouterFichier
		nm.ajouterFichier(f);

		// Test de GetInfoFichier
		assertEquals(f,nm.getInfoFichier("f.txt"));
	}
	@Test
	public void testEst_occupé_setStatus() throws UnknownHostException, AlreadyExists {
		NameNode nm = new NameNode();
		InfoFichier f = new InfoFichier("f.txt");
		f.addChunk(1, (Inet4Address)Inet4Address.getLocalHost());
		nm.ajouterFichier(f);

		// Test de setStatus
		nm.setStatus("f.txt", true);

		// Test de Est_occupé
		assertEquals(true,nm.est_occupé("f.txt"));
	}
	@Test
	public void testAddDataNode_getDataNodes_RemoveDataNode_estPresente() throws
UnknownHostException {
		NameNode nm = new NameNode();
		ArrayList<Inet4Address> a = new ArrayList<Inet4Address>();
		a.add((Inet4Address)Inet4Address.getLocalHost());
		ArrayList<Inet4Address> vide = new ArrayList<Inet4Address>();

		// Test AddDataNode et estPresente
		nm.addDataNode((Inet4Address)Inet4Address.getLocalHost());
		assertEquals(true,nm.estPresente((Inet4Address)Inet4Address.getLocalHost()));

		// Test de getDataNodes
		assertEquals(a,nm.getDataNodes());

		// Test de removeDataNodes et estPresente
		nm.removeDataNode((Inet4Address)Inet4Address.getLocalHost());

	assertEquals(false,nm.estPresente((Inet4Address)Inet4Address.getLocalHost()));

	}

}
```

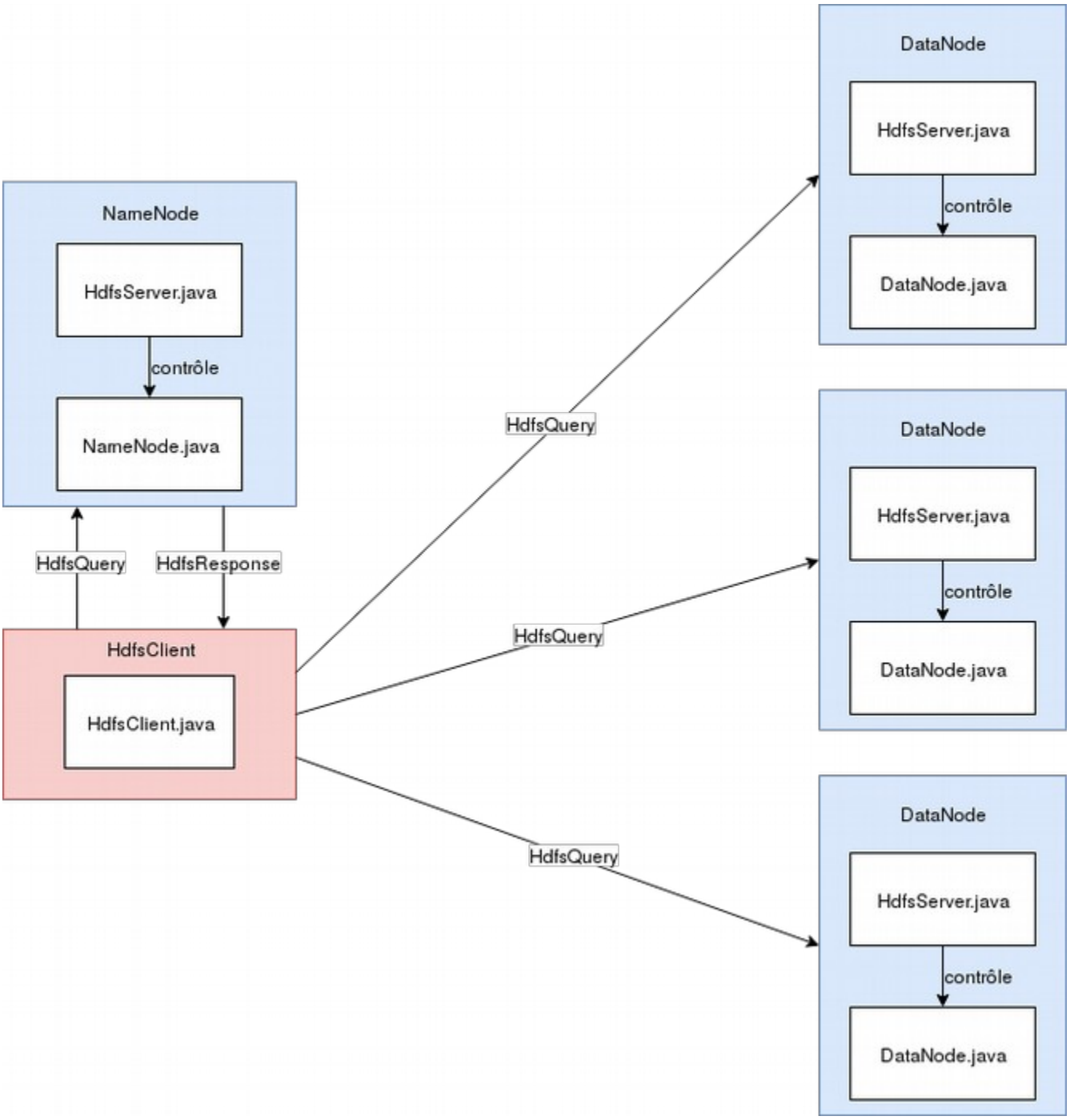

Figure 1: Architecture du service Hdfs implanté

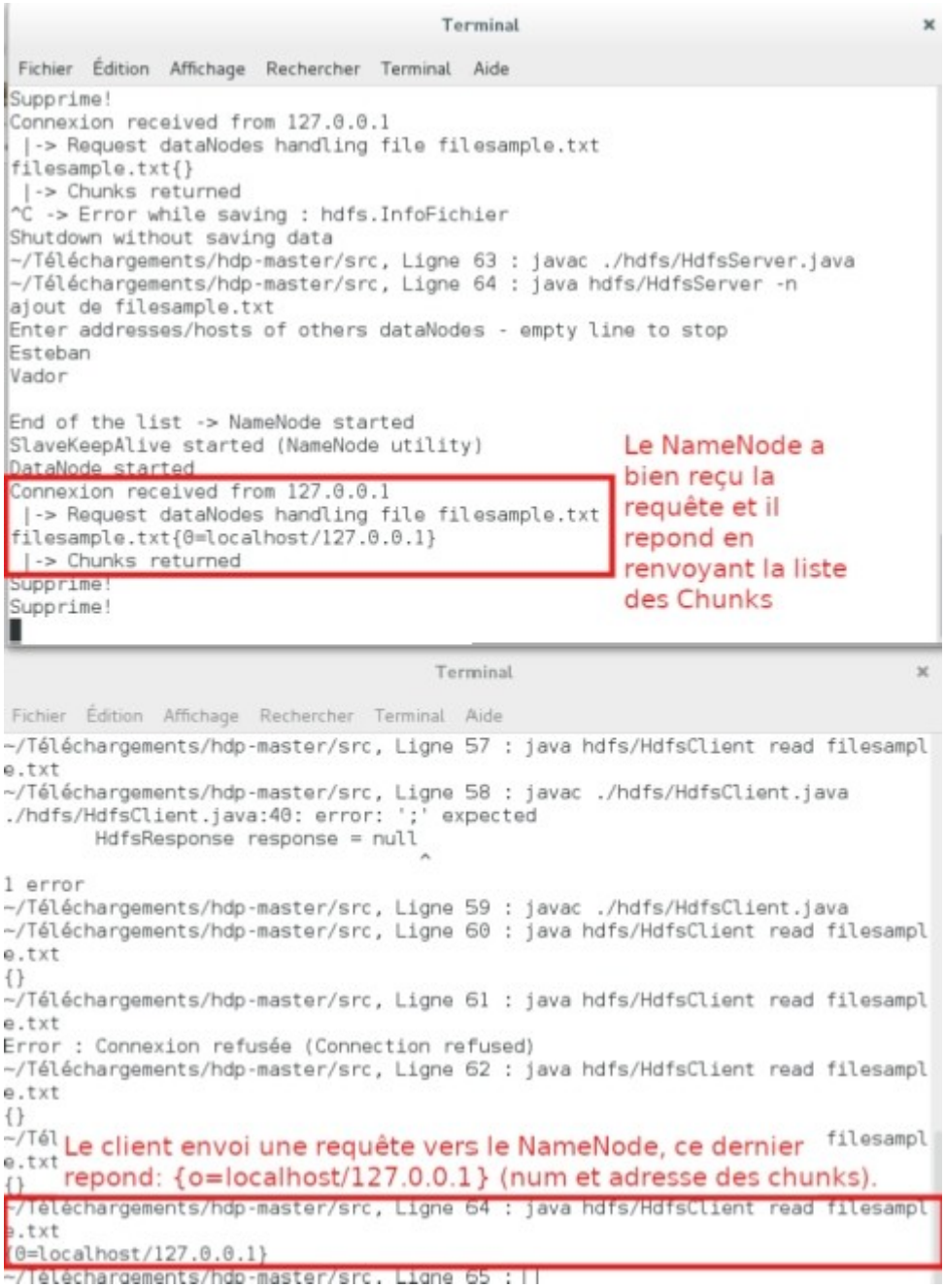

Figure 2: Test de communication entre HdfsClient et NameNode

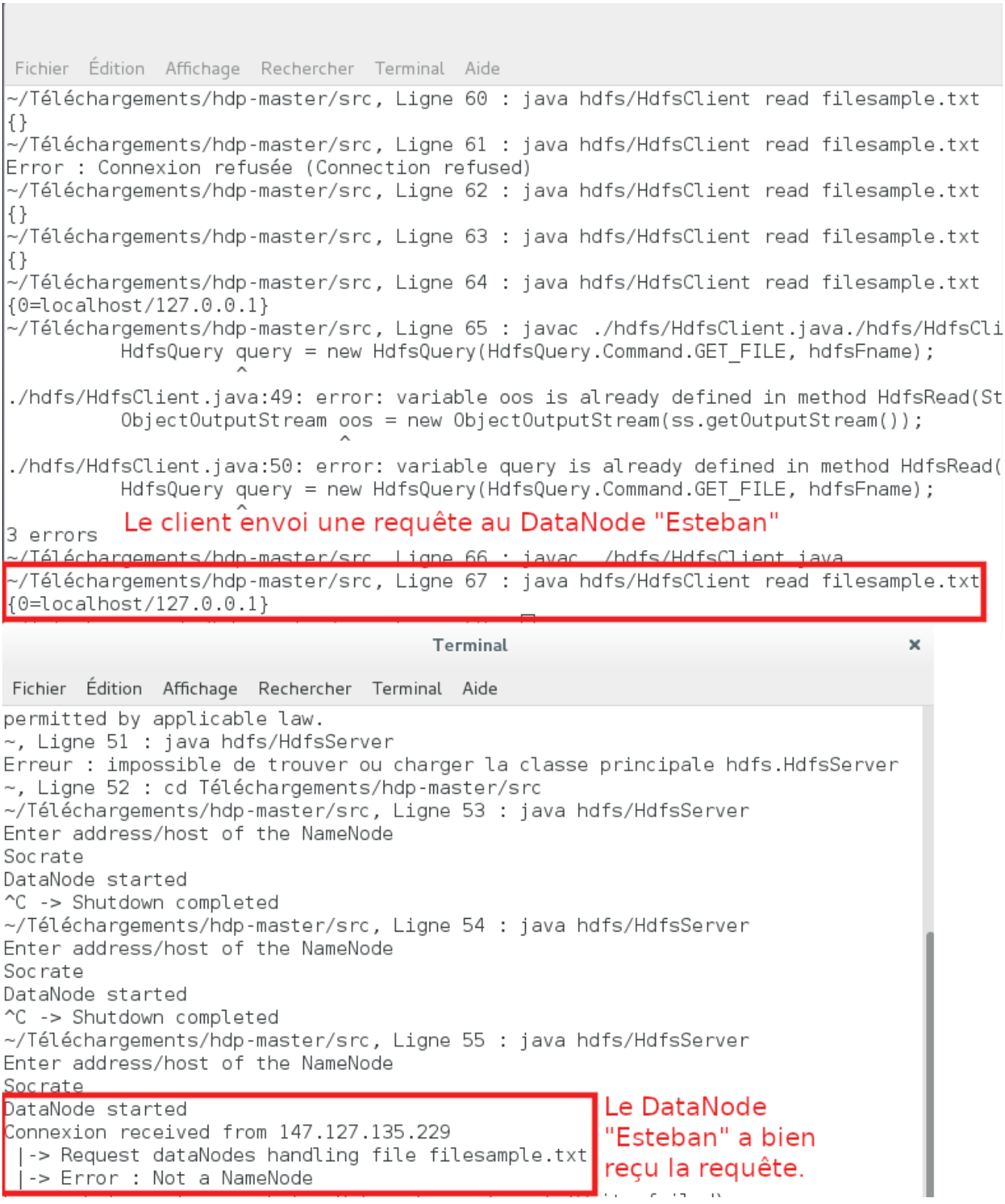

Figure 3: Communication entre HdfsClient et un DataNode de hostname "Esteban"