University of Southampton

COMP2207 2022/23
Distributed Systems and Networks

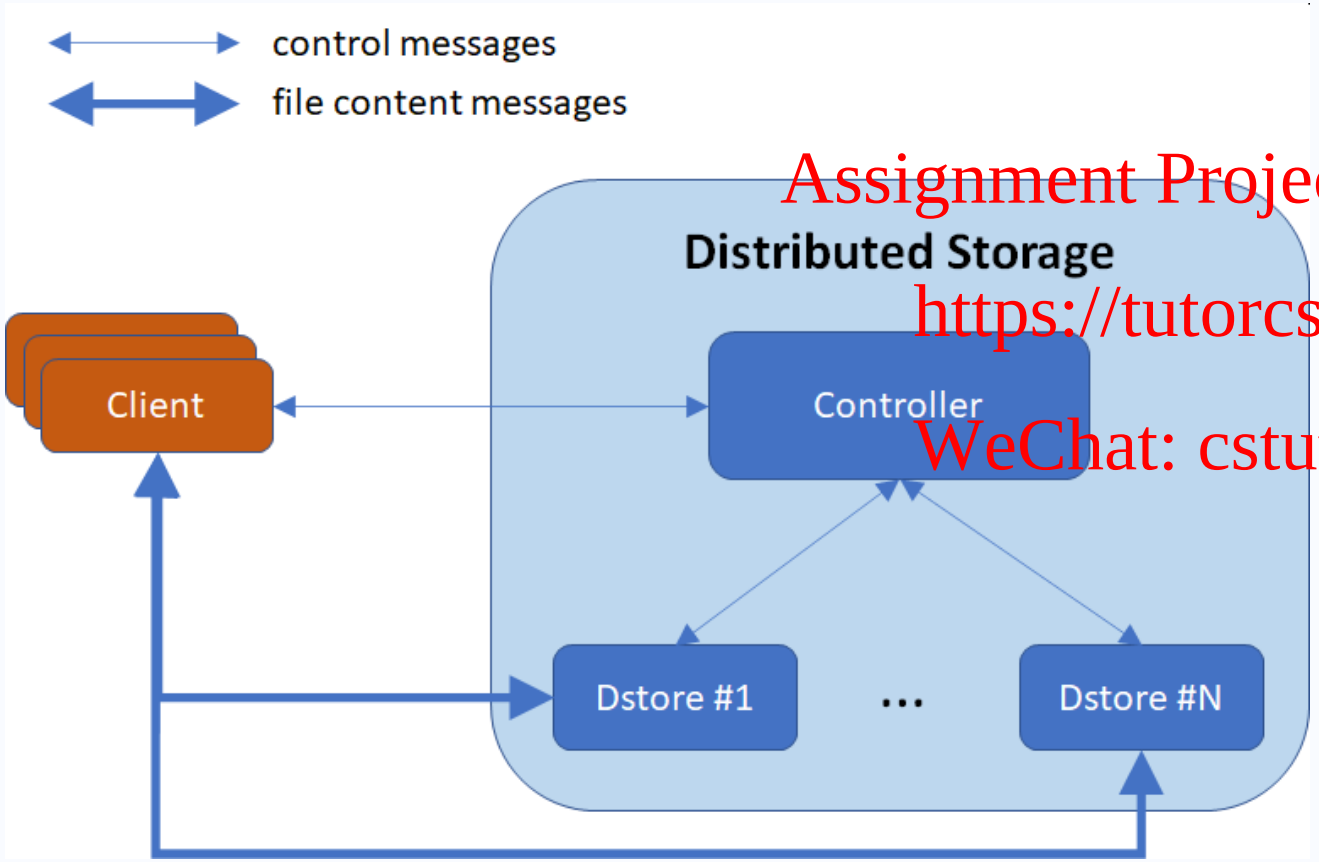
Coursework

Dr Leonardo Aniello
l.aniello@soton.ac.uk

Session Outline

- **Coursework specification**
- How to use the Client
- Recommendations
- Support

Coursework specification



- One Controller and N Data Stores (Dstores)
- Supports multiple concurrent clients sending **store**, **load**, **list**, **remove** requests
- You will implement Controller and Dstores
  - The client will be provided
- Each file is replicated R times over different Dstores
- Files are stored by the Dstores
- The Controller orchestrates client requests and maintains an index with the allocation of files to Dstores
- Files in the distributed storage are not organised in folders and sub-folders
  - Filenames do not contain spaces
- As Dstores may fail and new Dstores can join the storage system at runtime, rebalance operations are required to make sure each file is replicated R times and files are distributed evenly over the Dstores

Coursework specification

Networking

- Controller, Dstores and Clients will communicate with each other via TCP connections
- The Dstores will establish connections with the Controller as soon as they start.
  - These connections will be persistent
  - All the communications between a Dstore and the Controller must take place over that connection; no further connections can be established between a Dstore and the Controller
  - If the Controller detects that the connection with one of the Dstores dropped, then such a Dstore will be removed from the set of Dstores that are part of the storage system
- Send textual messages using the println() method of PrintWriter class
- Receive textual messages using the readLine() method of BufferedReader class
- Send data messages using the write() method of OutputStream class
- Receive data messages using the readNBytes() method of InputStream class

Coursework specification

The Index

- It refers to the data structure used by the Controller to keep track of files
  - Used to ensure that other possibly conflicting concurrent operations are served properly
- Example: while a file F is being stored
  - Serve any Load, Remove, List operations on F as if F did not exist
  - But serve any Store operation on F as it F already existed
- The coursework specification includes a section that defines how concurrent store and remove operations on a same file should be handled

Coursework specification

Code development

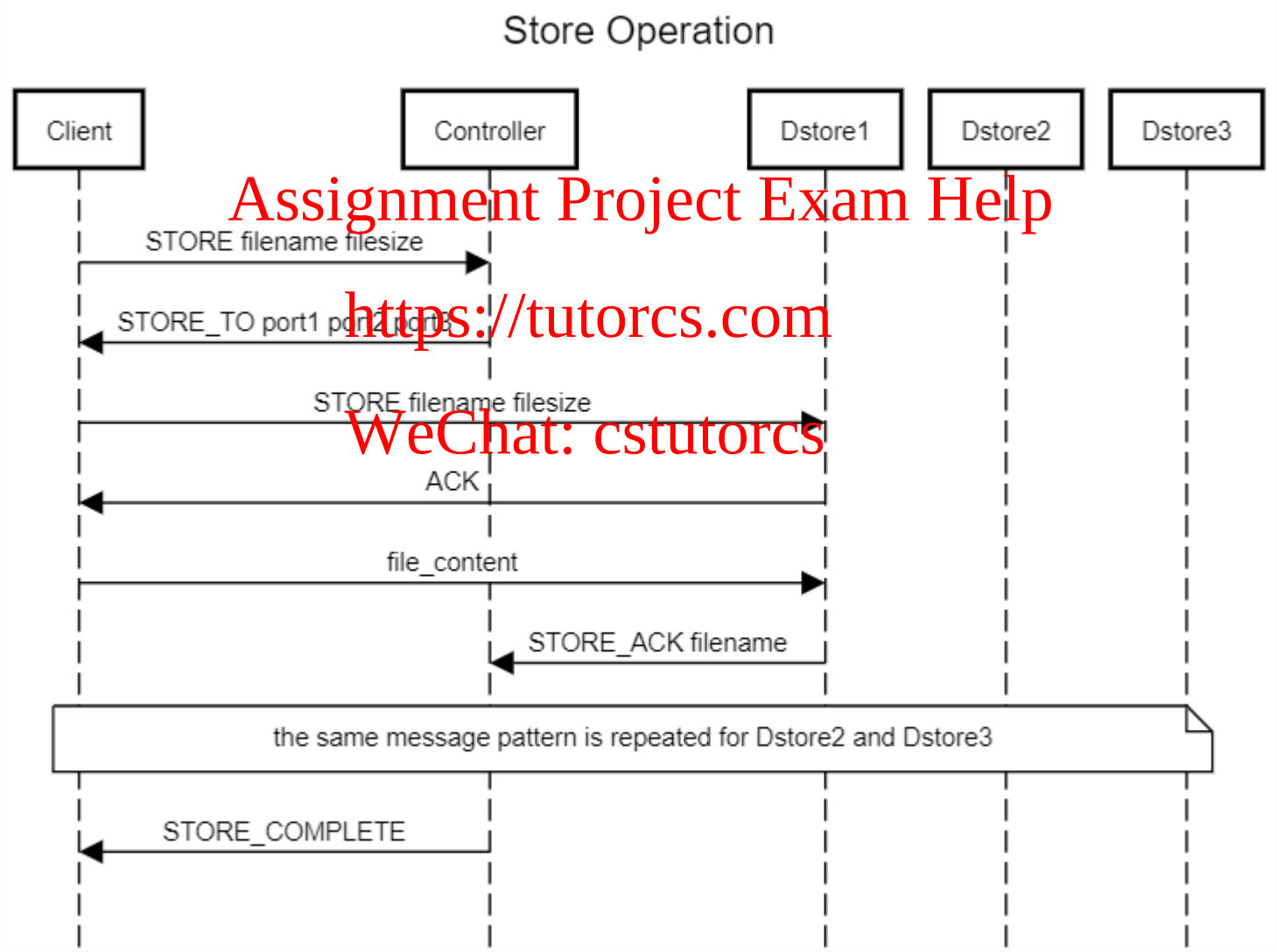
- Your code will be assessed using Java openjdk-17-jdk on Ubuntu 20 (or 22)
  - It is important that you test your code via command line using the same platform
- Command line parameters to start up the system
  - Controller: `java Controller cport R timeout rebalance_period`
  - A Dstore: `java Dstore port cport timeout file_folder`
  - A client: `java Client cport timeout`

Coursework specification

Code development

- Operations
  - Store
  - Load
  - Remove
  - List
  - Rebalance

Coursework specification



Coursework specification

Code development – Store operation

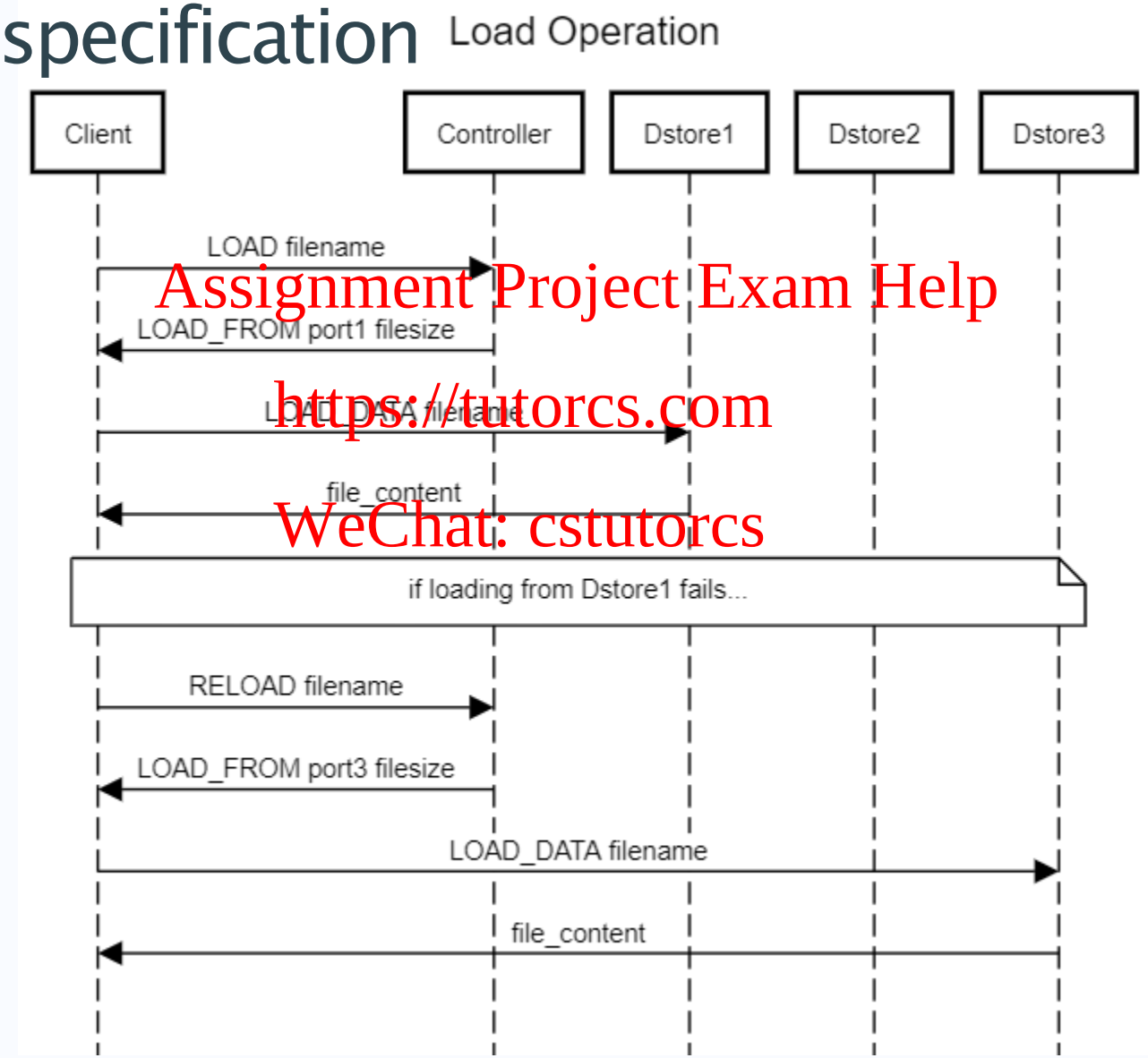
- Client -> Controller: `STORE filename filesize`
- Controller
  - *updates index, "store in progress"*
  - *Selects R Dstores, their endpoints and port1, port2, …, portR*
  - Controller -> Client: `STORE_TO port1 port2 … portR`
- For each Dstore i
  - Client->Dstore i: `STORE filename filesize`
  - Dstore i -> Client: `ACK`
  - Client->Dstore i: `file_content`
  - *Once Dstore i finishes storing the file*, Dstore i -> Controller: `STORE_ACK filename`
- Once Controller received all acks
  - *updates index, "store complete"*
  - Controller -> Client: `STORE_COMPLETE`

Coursework specification

Code development – Store operation – Failure Handling

- Malformed message received by Controller/Client/Dstore
  - Ignore message (it would be good practice to log it)
- If not enough Dstores have joined
  - Controller->Client: `ERROR_NOT_ENOUGH_DSTORES`
- If filename already exists in the index
  - Controller->Client: `ERROR_FILE_ALREADY_EXISTS`
- Client cannot connect or send data to all R Dstores
  - No further action, the state of the file in the index will remain "store in progress"; future rebalances will try to sort things out by ensuring the file is replicated to R Dstores
- If the Controller does not receive all the acks (e.g., because the timeout expires), the `STORE_COMPLETE` message should not be sent to the Client, and filename should be removed from the index

Coursework specification



Coursework specification

Code development – Load operation

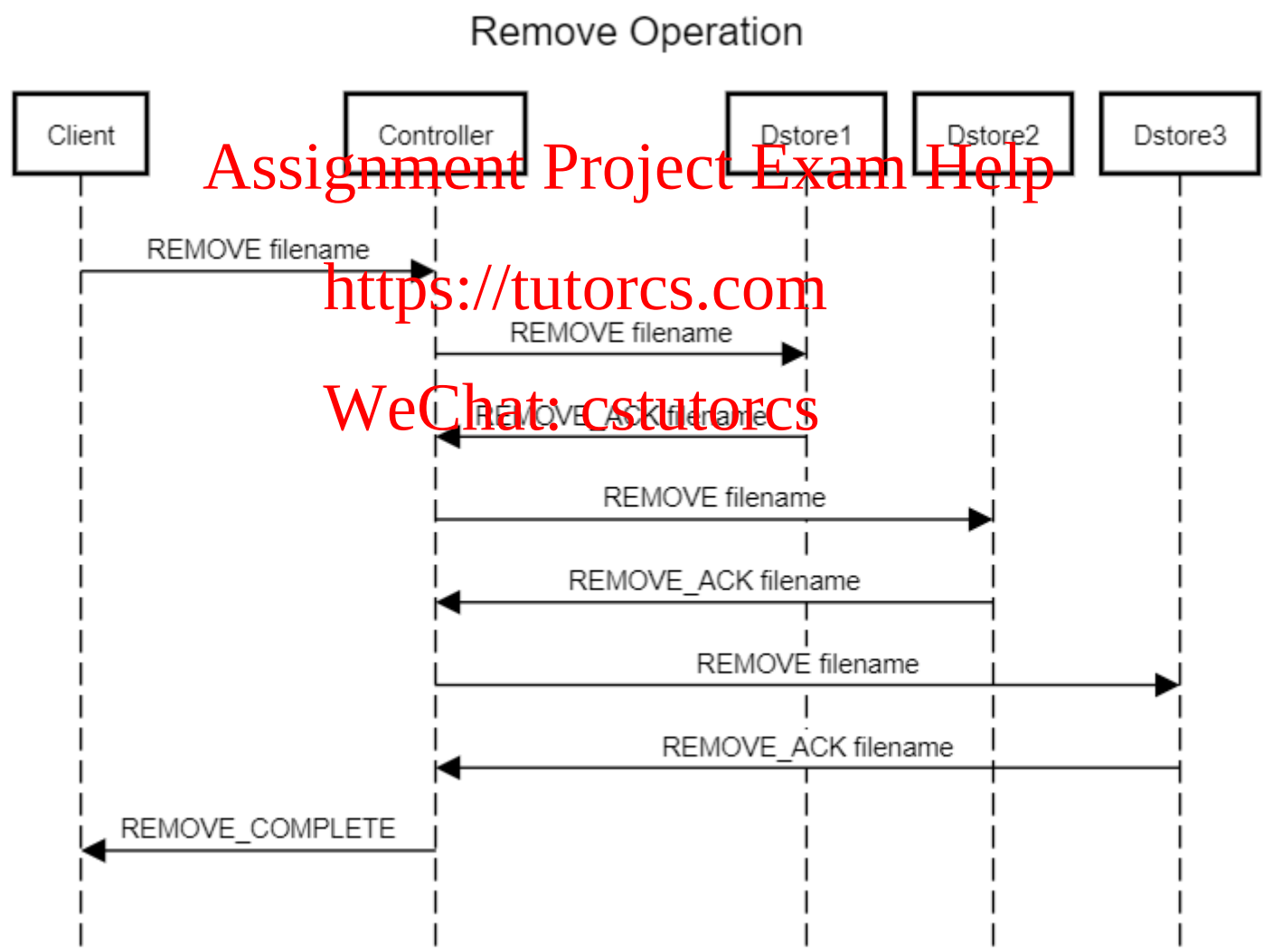
- Client -> Controller: `LOAD filename`
- *Controller selects one the R Dstores that stores that file, let port be its endpoint*
- Controller->Client: `LOAD_FROM port filesize`
- Client -> Dstore: `LOAD_DATA filename`
- Dstore -> Client: `file_content`

Coursework specification

Code development – Load operation – Failure Handling

- Malformed message received by Controller/Client/Dstore
  - Ignore message (it would be good practice to log it)
- If not enough Dstores have joined
  - Controller->Client: `ERROR_NOT_ENOUGH_DSTORES`
- If file does not exist in the index
  - Controller -> Client: `ERROR_FILE_DOES_NOT_EXIST`
- If Client cannot connect to or receive data from Dstore
  - Client -> Controller: `RELOAD filename`
  - *Controller selects a **different** Dstore with endpoint port'*
  - Controller->Client: `LOAD_FROM port' filesize`
  - *If Client cannot connect to or receive data from any of the R Dstores*
    - Controller->Client: `ERROR_LOAD`
- *If Dstore does not have the requested file, simply close the socket with the Client*

Coursework specification



Coursework specification

Code development – Remove operation

- Client -> Controller: `REMOVE filename`
- *Controller updates index, "remove in progress"*
- *For each Dstore i storing filename*
  - Controller->Dstore i: `REMOVE filename`
  - *Once Dstore i finishes removing the file*, Dstore i -> Controller: `REMOVE_ACK filename`
- *Once Controller received all acks*
  - *updates index, "remove complete"*
  - Controller -> Client: `REMOVE_COMPLETE`

Coursework specification

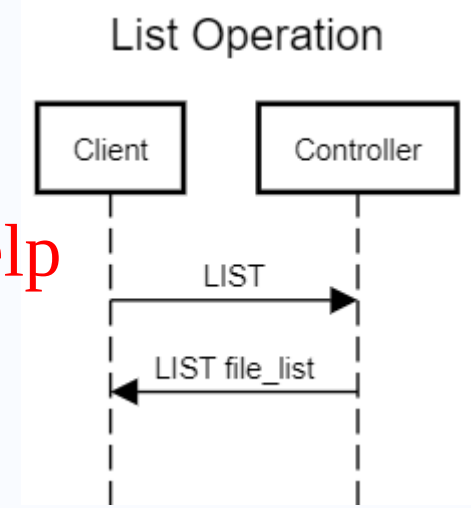
Code development – Remove operation – Failure Handling

- Malformed message received by Controller/Client/Dstore
  - Ignore message (it would be good practice to log it)
- If not enough Dstores have joined
  - Controller->Client: `ERROR_NOT_ENOUGH_DSTORES`
- If filename does not exist in the index
  - Controller->Client: `ERROR_FILE_DOES_NOT_EXIST`
- Controller cannot connect to some Dstore, or does not receive all the ACKs within the timeout
  - No further action, the state of the file in the index will remain "remove in progress"; future rebalances will try to sort things out by ensuring that no Dstore stores that file
- If Dstore does not have the requested file
  - Dstore -> Controller: `ERROR_FILE_DOES_NOT_EXIST filename`

Coursework specification

Code development – List operation

- Client->Controller: `LIST`
- Controller->Client: `LIST file_list`
  - *file_list is a space-separated list of filenames*



Failure Handling

- Malformed message received by Controller/Client
  - Ignore message (it would be good practice to log it)
- If not enough Dstores have joined
  - Controller->Client: `ERROR_NOT_ENOUGH_DSTORES`

Coursework specification

Code development – Rebalance operation

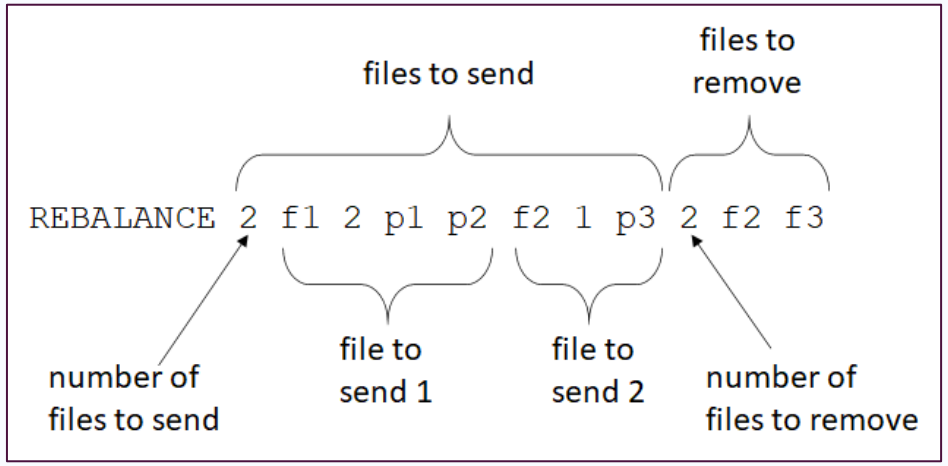
- This operation is started periodically by the Controller (i.e., based on the `rebalance_period` argument) and when a new Dstore joins the storage system
  - In the latter case, this is the message Dstore -> Controller: `JOIN port`
    - Where port is the endpoint of the new Dstore
- Each rebalance operation consists of 4 steps
  1. Controller asks each Dstore what files it stores
  2. Controller revises file allocation
  3. Controller tells each Dstore which files it should send to other Dstores or remove
  4. Each Dstore sends/removes specified files and inform Controller once finished

Coursework specification

1. Controller asks each Dstore what files it stores
   - Controller -> Dstore i: `LIST`
   - Dstore i -> Controller: `LIST file_list`
     - file_list is a space-separated list of filenames
2. Controller revises file allocation to ensure
   - Each file is replicated over R Dstores
   - Files are evenly stored among Dstores
     - With N Dstores, replication factor R, and F files, each Dstore should store between floor(RF/N) and ceil(RF/N) files
     - E.g., with N=7, R=3, F=10, each Dstore should store between 4 and 5 files

Coursework specification

3. Controller tells each Dstore which files it should send to other Dstores or remove
   - Controller produces for each Dstore i a pair (`files_to_send`, `files_to_remove`), where
     - `files_to_send` is the list of files to send and is in the form `number_of_files_to_send file_to_send_1 file_to_send_2 … file_to_send_N`
       - `file_to_send_i` is in the form `filename number_of_dstores dstore1 dstore2 … dstoreM`
     - `files_to_remove` is the list of filenames to remove and is in the form `number_of_files_to_remove filename1 filename2 … filename`
   - Controller->Dstore i: `REBALANCE files_to_send files_to_remove`
   - Example:
     - Assume that
       - file f1 needs to be sent to Dstores p1 and p2
       - file f2 needs to be sent to Dstore p3
       - file f2 needs to be removed
       - file f3 needs to be removed
     - `REBALANCE 2 f1 2 p1 p2 f2 1 p3 2 f2 f3`



Coursework specification

4. Each Dstore sends/removes specified files and inform Controller once finished
   - Dstore i will send required files to other Dstores; e.g., to send a file to Dstore j
     - Dstore i -> Dstore j: `REBALANCE_STORE filename filesize`
     - Dstore j -> Dstore i: `ACK`
     - Dstore i -> Dstore j: `file_content`
   - *Dstore i will remove specified files*
   - When rebalance is completed
     - Dstore i -> Controller: `REBALANCE_COMPLETE`

Coursework specification

Code development – Rebalance operation – Failure Handling

- Malformed message received by Controller/Dstore
  - Ignore message (it would be good practice to log it)
- Controller does not receive REBALANCE_COMPLETE from a Dstore within a timeout
  - No further action; future rebalance operations will sort things out

Additional notes on Rebalance operations

- The first rebalance must start `rebalance_period` seconds after the Controller started
- Clients' requests are queued by the Controller during rebalance operations; these requests will be served once the rebalance operation is completed
- A rebalance operation should wait for any pending STORE and REMOVE operation to complete before starting
- Dstores will not be terminated during this operation (but might fail)
- If it turns out that the index includes a file that no Dstore included in the list sent to the Controller, then it would be safe to remove this file from the index

Coursework specification

Submission Requirements

- **Submissions Deadline: Thursday 18 May**
- Your submission should include the following files:
  - Controller.java
  - Dstore.java
  - As well as all the additional .java files you developed
    - These include `Protocol.java` if you used it
- These files should be contained in a single zip file called <your username>.zip
  - There should be no package structure to your java code
  - When extracted from the zip file, the files should be located in the current directory
- These files will be executed at the Linux command line by us for automatic testing

Coursework specification

Marking Scheme

- Up to 50 marks are awarded based on whether the storage system works in compliance with the protocol and correctly serves sequential requests from a single client
- Up to 10 marks are awarded based on whether each file is replicated R times and files are evenly spread over the Dstores (only when stored, not when Dstores fail or new Dstores join the storage system)
- Up to 10 marks are awarded based on whether the storage system correctly serves concurrent requests from more clients (up to 10 concurrent clients)
- Up to 10 marks are awarded based on whether the storage system correctly tolerates the failure of one Dstore
- Up to 10 marks are awarded based on whether the storage system correctly tolerates the failure of up to N-R Dstores
- Up to 10 marks are awarded based on whether files are evenly spread over the Dstores despite Dstores failing and new Dstores joining the storage system

Session Outline

- Coursework specification
- ➢ **How to use the Client**
- Recommendations
- Support

How to use the Client

- Download zip from the wiki
  - `client.jar` – obfuscated library
  - `ClientMain.java` – example of how to use the library, you are expected to customise it
- Compilation
  - Make sure `client.jar` and `ClientMain.java` are in the current directory
  - From terminal, type: `javac -cp client.jar ClientMain.java`
- Execution
  - On Linux - from terminal, type: `java -cp client.jar:. ClientMain 12345 1000`
  - On Windows - from terminal, type: `java -cp client.jar;. ClientMain 12345 1000`

How to use the Client

- Client class
  - `public Client(int cport, int timeout, Logger.LoggingType loggintType)`
  - `public void connect() throws IOException`
  - `public void disconnect() throws IOException`
  - `public String[] list() throws IOException, NotEnoughDstoresException`
  - `public void store(File file) throws IOException, NotEnoughDstoresException, FileAlreadyExistsException`
  - `public void store(String filename, byte[] data) throws IOException, NotEnoughDstoresException, FileAlreadyExistsException`
  - `public void load(String filename, File fileFolder) throws IOException, NotEnoughDstoresException, FileDoesNotExistException`
  - `public byte[] load(String filename) throws IOException, NotEnoughDstoresException, FileDoesNotExistException`
  - `public void remove(String filename) throws IOException, NotEnoughDstoresException, FileDoesNotExistException`

How to use the Client

- Client class – other useful methods
  - `public void **send**(String message)`
    - It sends a custom String message to the Controller
  - `public byte[] **wrongLoad**(String filename, int howManyDstoresToContactAtMost) throws IOException, NotEnoughDstoresException, FileDoesNotExistException`
    - Executes the Load operation but does not actually load the file from any Dstore, it keeps sending RELOAD messages to the Controller
  - `public void **wrongStore**(String filename, byte data[]) throws IOException, NotEnoughDstoresException, FileAlreadyExistsException`
    - Executes the Store operation but do not send the file to any Dstore

Session Outline

- Coursework specification
- How to use the Client
- ➢ **Recommendations**
- Support

Recommendations

Timeouts

- Only use timeouts when a process expects a response from another process
- See `Socket.setSoTimeout()` method

Waiting for a number of ACKs

- For some operations, the Controller needs to wait to receive ACK messages from R distinct Dstores
- see `CountDownLatch` class

Revise Multi-threading and Synchronisation from COMP2106

Session Outline

- Coursework specification
- How to use the Client
- Recommendations
- ➢ **Support**

Support Available

- ECS Programming Support service
  - General help with programming
  - Discord: discord.ecs.soton.ac.uk – look for the Helpdesk channel
  - Email: helpdesk@ecs.soton.ac.uk
- Coursework channel in COMP2207 group on Teams
- Please do not send me messages on Teams, use email instead
- Office hours on Fridays 3-4pm in 59/3221 (during term time)
- Two coursework surgery sessions
  - Thursday 27 April 9am
  - Thursday 11 May 9am
- Script to validate your submission locally – **coming soon**
- Service to let you upload your submission and run some predefined tests in the target environment – **coming soon**