DHT Second iteration - Group 2

Compiling program

GUI (written in Java) and node (written in C) have to be compiled separately. Node can be compiled using the makefile in `code/c` which puts its results (object files and an executable) into `code/c/build`. GUI can be compiled using the makefile in `code/java`, which similarly puts its results (class files and a jar) into `code/java/build`. Tested compilers are gcc 4.6 & 4.8 and javac 1.6.

Using the program

After both the GUI and node have been compiled they have to be started separately. Node can be started with command `./dhtnode <options>` and it has several options to modify its behaviour. Options are listed in a table below.

Option	Short form	Long form	Default
Host address	-A	--hostaddr	localhost
Host port	-P	--hostport	2000
Server address	-a	--servaddr	localhost
Server port	-p	--servport	1234
Block directory	-b	--blockdir	blocks
Logging level	-l	--loglevel	3

Host address and port:
Address and port of the computer where the program is run. When all nodes and the server are located on a single computer localhost can be used. However, when connections are to be made between computers all nodes should specify their proper address using -A in addition to server's address.

Server address and port:
Address and port of the computer where the server is running

Block directory:
Directory where maintained blocks are stored

Logging level:
Logging messages of this level and below are printed. Levels are as follows:

- No log = 0
- Error = 1
- Warn = 2
- Info = 3
- Debug = 4

GUI can be started with command `java -jar DHT.jar`. After GUI has been started the first thing to do is to connect to a node. This can be done by clicking “connect”, filling out required information and then clicking OK. After connection has been established files can be putted, getted and dumped using the respective buttons. Files can be also getted and dumped by right-clicking files in the directory view.

Implementation

Advanced features we would like to be assessed

- Directory
- Multi-block files

(Progress bar is also fully implemented because it is cool)

Maintaining, fetching and storing data

For maintaining data blocks dhtnode has a keyring data structure. Keyring is a circular doubly-linked list where each item is a SHA1 key. When keyring is created it is initialized with node’s host key which then serves as an entry point to the ring. When keys are added to the ring they are inserted to the proper position (from smallest to largest) relative to the host key and other ring. In addition to adding, deleting and searching keys the keyring supports slicing. Slices between arbitrary keys (they don’t actually have to be in the ring) can be removed from the ring.

Whenever a node receives a data block (either from put request or transfer-on-join) the block’s key is added to the keyring created when node is first started. The data itself is written on disk. When node receives get or dump requests it first checks if it has the key and reacts accordingly (e.g. send data if it exists or NO_DATA if not). When other node joins the existing node slices the ring between the midpoint keys between existing node’s key and joining node’s key. Data in this slice is then sent to the joining node. Similarly, when a node leaves the DHT two slices are created: one for the left neighbour and one for the right. During the disconnection sequence these slices are sent to neighbours.

While node expects that for every key in keyring there is a corresponding file (except for host key) if it fails to read the said file (for example because the user has deleted it) it simply deletes the key and responds that no data was found. Node handles most errors in a similar way: it simply logs an error message and proceeds to wait for next packet/command.

Transferring files

Most of the work during file transfer is done in Java and node simply relays commands from GUI to server. Sometimes this results in unnecessary traffic (for example, when users requests file that is maintained by the user's node) but made node's design a lot simpler. Like in previous iteration, node is still mostly stateless.

DHT Controller

DHTController is the core java class that implements the functionality of the DHT. It connects to a node when initiated and handles all communication with the node through NodeIO class. DHTController offers public methods putFile(), getFile(), dumpFile() and terminate() which allow the user interface to call these operations. They handle dividing and combining files into blocks (max 65531 bytes, 4 bytes are needed for block numbering) which they pass to the node and get from the node. The blocks are named <fileName>-PARTX so the first block is always named <fileName>-PART1 and it represents the entire file in the DHT. If the first block of the file is missing then the file can not be found.

Controller / Node communication

DataBlock class builds messages that we call commands. Commands form the protocol for Controller / Node communication. Basically Controller sends commands like CMD_PUT_DATA and always waits for node's response. If the response is CMD_PUT_DATA_ACK put was successful otherwise put failed and the operation is aborted. Communication between Controller and Node is similar to communication between the server and Node.

GUI

GUI is made by using Swing, Java GUI widget toolkit. It has following basic features:

- Connect to node
- Disconnect node from server
- Put data to DHT
- Get data from DHT
- Dump data from DHT
- Exit from server normally or abnormally (as explained in instructions)
- DHT directory
- Log
- Refresh DHT directory
- Progress bar (though in its own class and file)

After starting GUI with `java -jar DHT.jar` command, GUI class does initialization for GUI frame, buttons, panels, basic variables, action listeners, upper menu, DHT directory and log.

GUI is used to connect to existing node. By pressing connect from upper menu (File -> Connect) or Connect on main screen, pop up dialog appears and asks user to give Host (node's) address and port. There are given default values 'localhost' and '2000'. If node exists, connection is made

by initializing DHTController with given values.

After connecting the user may put data to DHT, get data from DHT, dump data from DHT, refresh DHT directory and disconnect from the server. For every action there exists corresponding buttons in the main screen and in the upper menu (File -> …). In addition fetching and dumping data from DHT can be done through DHT directory window (explained further in Directory paragraph). Also in upper menu there is Exit button (File -> Exit) with which user may Exit from the program. If user is connected to node, pop up window asks whether he/she wants to leave the server ‘normally’ or ‘abnormally’ (as explained in instructions) or cancel exit operation. Pressing cross button makes node leave the server abnormally.

File selection for putting and getting data is provided by JFileChooser. When putting data, JFileChooser opens OpenDialog where user can graphically determine which file and from where he/she desires to put in DHT. After selecting the file new Dialog opens to ask which name the user likes to give for the file in DHT (defaults for the name of selected file). Then the given arguments are passed for DHTController, calling its method putFile. Same behaviour is for getting data (OpenDialog is SaveDialog and putFile is getFile), but vice versa.

When dumping a file, the user is asked for to give a name of the file in DHT to be dumped. After getting file name it is passed for DHTController and its method dumpFile is called.

When node wants to disconnect from the server, it calls for DHTControllers method terminate. Based on its return value, GUI performs and informs the user. These cases are, as explained in instructions.

If the user wants to exit program, he/she may use cross button or File -> Exit. From cross button program closes without further questions (leaves ‘abnormally’). From File -> Exit, the user is asked whether he/she wants to leave normally or abnormally. If the user chooses abnormally, program closes like from cross button. From normally closing, disconnect is performed and DHTController’s method terminate is called. If disconnect is denied or termination fails, program won’t shut and node won’t disconnect. Otherwise program will shut.

Log is JTextArea inserted in JScrollPane, to show stderr messages in the log window. This is totally extra functionality, but helps for debugging and understanding performance of the program. Log updates itself simultaneously with the processes (for example this can be seen while putting big file in DHT), but it doesn’t scroll down at the same time, though it updates scroll on the bottom when process is done.

Every action listener (Disconnect, Connect, Put, Get, Dump etc.), where actions are performed depending which action the user has launched, are GUI’s subclasses implementing Actionlistener.

Progress bar

Progress bar is displayed in its own class Progressbar in Progressbar.java. With initialization, new frame, panel and progress bar are created. Progress bar is updated via method update, where new progress value is given, new value is set to progress bar and new progress bar is 'painted'. If process faces unexpected ending and progress bar doesn't reach 100% (for example if node tries to disconnect when it's the only node, progress ends up at 67%), interrupt method is called, where progress bar is set invisible. In this case DHTController gives the return value of the process for GUI which informs the user why the process didn't reach 100%. When everything goes fine (progress bar reached 100%) return value is 0.

DHTController controls progressbar. When any action performed, DHTController initializes class Progressbar with values int init (first step of the progress), int maxValue (maximum value of the progress) and String status (the process under performance). When the Controller finds out how many blocks need to be transferred in order to complete the operation it initiates the progress bar. The progress bar is then incremented in other methods whenever a command is passed to or from the node.

Directory

Directory is JList showed in GUI in JScrollPane panel. The user may fetch or dump files by pressing right click on desired file, and selecting option from given pop up menu. Directory can be refreshed by refresh button beside it, and is always "refreshed" when actions performed by the user (for example after connecting, putting data, etc.)

DHTController controls directory. When GUI wants to refresh directory (after any action, refresh button pressed etc) it calls DHTController's method refreshDHTdir or getDHTdir, which passes String array for GUI's method directory. This method then converts String array to JList and creates new JScrollPane with JList inside it. The Controller manages the directory locally as a Linked list and keeps a block named "DHTDIR-PART1" where the DHT writes down a list of all known files in the directory. The directory is the only block that gets locked whenever it is handled by a controller and released after it is put back. Every time a file is added or dumped the block is altered accordingly.

Testing

Again, testing was done mostly in Paniikki and Niksula computers. We didn't have very rigid testing methods: we simply created a network (up to three computers) and performed operations (put, get, dump, disconnect) in the DHT. For the record, the biggest file put and gotten back from DHT was 37,7 megabytes (just took some time).

Known bugs and problems

- GUI may not be beautiful, but it's easy to use and it works
- GUI's components aren't scalable
- Only directory block is locked so problems may occur when files with the same name are handled simultaneously
    - As a rule of thumb, things work if there aren't multiple operations going on at the same time
- If blocks are lost from the network for some reason (like for example if a node dies) the files get corrupted and can't therefore be downloaded

Time management

We didn't really track how much time we spent on this iteration. We estimated that we spent around 30 hours on the first iteration and it is safe to say that we spent more time on this second iteration.

Teamwork

Implementation, design and testing

During the first iteration we worked mostly together on one computer so there weren't any issues when integrating our work. In this iteration, however, we worked differently. Although we still often worked together (as in next to each other) we divided our work a lot more. The different areas of responsibility were node functionality, main Java functionality and GUI. Design of the components was left mostly for the person responsible. We still discussed our decisions and especially integrating Java components required working together. Implementing functionality in node was mostly just implementing the specification.

Problems

Although dividing tasks allowed us to work more efficiently there were some problems. Quite often we edited same file simultaneously and this resulted in conflicts in SVN. Resolving these conflicts was for some reason very difficult and we often had to delete whole working copy and checkout again. Luckily these problems were mostly just an annoyance.

Other problem was that we (or mostly just one of us) sometimes failed to communicate the changes we had made. Usually this wasn't a big deal but resulted in confusion why some things behaved differently than they had done the day before.