Distributed Systems
COMP90015 2017 SM1
Project 1 - EZShare
Resource Sharing Network

Introduction

Groups should be able to connect their servers to others' clients, and vice versa.

In Project 1 we will build a resource sharing network that consists of servers, which can communicate with each other, and clients which can communicate with the servers. The system will be called EZShare.

In typical usage, each user that wants to share files will start an EZShare server on the machine that contains the files. An EZShare client can be used to instruct the server to share the files.

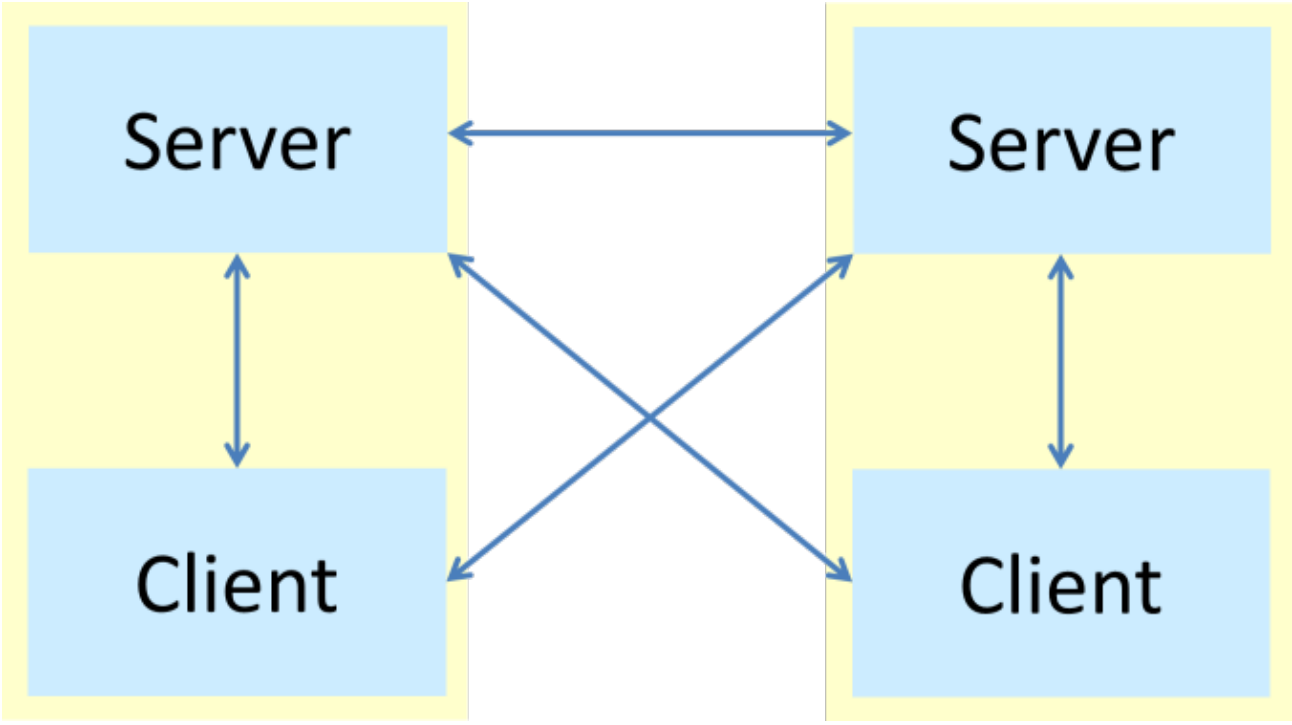
Servers can be queried for what files they are sharing. Clients can request a shared file be downloaded to them.

Servers can connect to other servers and queries can propogate throughout all of the servers.

In general, servers can publish resources; a file is just one kind of resource. In EZShare, other resources are just references (URIs) to e.g. web pages.

Every published resource (including shared files) has an optional owner and channel to which it belongs. These things allow resources to be controlled, e.g. not all shared resources have to be available to the public.

Architecture



Communication

All communication will be via TCP.

All messages, apart from file contents, will be in JSON format, one JSON message per line.

The text encoding for messages will be *Java Modified UTF-8 Encoding*, which is the format used by the **writeUTF()** and **readUTF()** methods in Java.

File contents will be transmitted as exact byte sequences, mixed between JSON messages as required.

Interactions will be synchronous request-reply, with a single request per connection.

Resource

Can use HashMap <String, Resource>
Concurrent HashMaps?
"Synchronized ..."?
Thread Pool model? A queue and multiple worker threads in a pool

A **Resource** has the following attributes:

- **Name**: optional user supplied name (String), default is "".
- **Description**: optional user supplied description (String), default is "".
- **Tags**: optional user supplied list of tags (Array of Strings), default is empty list.
- **URI**: mandatory user supplied absolute URI, that is unique for each resource on a given EZShare Server within each **Channel** on the server (String). The **URI** must conform to official URI format.
- **Channel**: optional user supplied channel name (String), default is "".
- **Owner**: optional user supplied owner name (String), default is "".
- **EZserver**: system supplied server:port name that lists the **Resource** (String).

Resources need to be stored, looked up and transmitted, so it will be wise to develop a robust **Resource** class.

Some special rules for strings are that they must not contain the null character `"\0"` and must not start or end with white space. The server may silently remove these things from receive resource descriptions. As well, the **Owner** cannot be the character "*".

Channels and Owners

Each resource *must* be stored and processed in a way that respects its **Channel** and **Owner**. You may think of the primary key for a resource being a tuple:

```
(owner,channel,uri)
```

The default **Owner** is "", the default **Channel** is "", and an absolute **URI** must always be given. The tuple becomes:

```
("","",uri)
```

A **Channel** may be used without an **Owner** and vice versa, and they may be used together.

There is no client command that will list used channels or owners. These things are kept secret by the server. The user needs to remember the owner and channel, if one was used, in order to refer to the resource at a later

time. The default channel can be thought of as the *public* channel. Other channels are thought of as *private* channels. The default owner means that anyone can update the resource. Otherwise updates require the correct owner name to work.

Shared File

A shared file is a **Resource** with a file **URI**, e.g. `file:///path/to/file.doc`.

The EZShare Server that lists the shared file can be asked to download it.

Other **URI**s, such as e.g. http and ftp, will be for informational purposes only. Accessing and downloading these is not required in this project.

EZShare Server Commands

- **PUBLISH**: create a new **Resource** and make it available
- **REMOVE**: remove an existing **Resource**
- **SHARE**: create a new **Resource** with a file **URI** and make it avialble
- **QUERY**: list all resources that match a **Resource** template
- **FETCH**: download all resources that match a **Resource** template which includes a file **URI**
- **EXCHANGE**: receive a list of EZShare host:port names

PUBLISH command

```
{
    "command": "PUBLISH",
    "resource": {
        "name": "Unimelb website",
        "tags": [
            "web",
            "html"
        ],
        "description": "The main page for the University of Melbourne",
        "uri": "http:\/\/www.unimelb.edu.au",
        "channel": "",
        "owner": "",
        "ezserver": null
    }
}
```

This creates a resource on the server with primary key `("","",http://www.unimelb.edu.au)`

PUBLISH rules enforced by server

- The command field is case sensitive (*this is the same for all commands*).
- A valid resource must be given. Missing fields may be filled in with defaults, which are the empty sting "" in most case, or the empty array for tags (*this is the same for all commands*).
- The URI must be present, must be absolute and cannot be a file scheme.
- Publishing a resource with the same primary key as an existing resource simply overwrites the existing resource.

- Publishing a resource with the same channel and URI but different owner is not allowed. This ensures that in any given channel, a given URI is only present once.
- String values must not contain the "\0" character, nor start or end with whitespace. The server may silently remove such characters or may consider the resource invalid if such things are found (*this is the same for all commands*).
- The **Owner** field must not be the single character "*". The resource is invalid in this case. (*This is the same for all commands.*)

PUBLISH responses from server

For a successful publish:

```
{ "response" : "success" }
```

If the publishing rules (other than below) were broken:

```
{ "response" : "error",
  "errorMessage" : "cannot publish resource"
}
```

If the resource contained incorrect information that could not be recovered from:

```
{ "response" : "error",
  "errorMessage" : "invalid resource"
}
```

If the resource field was not given or not of the correct type:

```
{ "response" : "error",
  "errorMessage" : "missing resource"
}
```

Generic responses from server

If the command is invalid (unknown):

```
{ "response" : "error",
  "errorMessage" : "invalid command"
}
```

If the command is missing or incorrect type:

```
{ "response" : "error",
  "errorMessage" : "missing or incorrect type for command"
}
```

REMOVE command

```
{
    "command": "REMOVE",
    "resource": {
        "name": "",
        "tags": [],
```

```
        "description": "",
        "uri": "http:\/\/www.unimelb.edu.au",
        "channel": "",
        "owner": "",
        "ezserver": null
    }
}
```

This will remove the resource with the primary key `("","",http://www.unimelb.edu.au)`.

The other fields of the resource are not needed, since only the primary key fields are required to remove the resource. If the other fields are given, they are ignored.

REMOVE responses from server

For a successful remove:

```
{ "response" : "success" }
```

If the resource did not exist:

```
{ "response" : "error",
  "errorMessage" : "cannot remove resource"
}
```

If the resource contained incorrect information that could not be recovered from:

```
{ "response" : "error",
  "errorMessage" : "invalid resource"
}
```

If the resource field was not given or not of the correct type:

```
{ "response" : "error",
  "errorMessage" : "missing resource"
}
```

SHARE command

```
{
    "command": "SHARE",
    "secret": "2os41f58vkd9e1q4ua6ov5emlv",
    "resource": {
        "name": "EZShare JAR",
        "tags": [
            "jar"
        ],
        "description": "The jar file for EZShare. Use with caution.",
        "uri":"file:\/\/\/\/home\/aaron\/EZShare\/ezshare.jar",
        "channel": "my_private_channel",
        "owner": "aaron010",
        "ezserver": null
    }
}
```

The SHARE command works almost identically to the PUBLISH command, with the major difference being that the URI must be a file scheme, while the PUBLISH command enforces that the URI cannot be a file scheme.

Another difference is that the server secret is required for the command to be successful.

SHARE rules enforced by the server

- The server secret must be present and must equal the value known to the server, for the command to succeed.
- The URI must be present, must be absolute, non-authoritative and must be a file scheme. It must point to a file on the local file system that the server can read as a file.
- Sharing a resource with the same primary key as an existing resource simply overwrites the existing resource (same as PUBLISH command).
- Sharing a resource with the same channel and URI but different owner is not allowed. This ensures that in any given channel, a given URI is only present once (same as PUBLISH command).

SHARE responses from server

For a successful share:

```
{ "response" : "success" }
```

If the rules (other than below) are broken:

```
{ "response" : "error",
  "errorMessage" : "cannot share resource"
}
```

If the resource contained incorrect information that could not be recovered from:

```
{ "response" : "error",
  "errorMessage" : "invalid resource"
}
```

If the secret was incorrect:

```
{ "response" : "error",
  "errorMessage" : "incorrect secret"
}
```

If the resource or secret field was not given or not of the correct type:

```
{ "response" : "error",
  "errorMessage" : "missing resource and\/or secret"
}
```

QUERY command

```
{
    "command": "QUERY",
    "relay": true,
```

```
    "resourceTemplate": {
        "name": "",
        "tags": [],
        "description": "",
        "uri": "",
        "channel": "",
        "owner": "",
        "ezserver": null
    }
}
```

This command also contains a **relay** field. In usual circumstances this would be set `true` by the client.

There is no resource, but rather a **resourceTemplate**. The purpose of the template is to specify the query in terms of desired fields that must match.

QUERY rules enforced by the server

The purpose of the query is to match the template against existing resources. The template will match a candidate resource if:

(The template channel equals (case sensitive) the resource channel **AND**

If the template contains an owner that is not "", then the candidate owner must equal it (case sensitive) **AND**

Any tags present in the template also are present in the candidate (case insensitive) **AND**

If the template contains a URI then the candidate URI matches (case sensitive) **AND**

(The candidate name contains the template name as a substring (for non "" template name) **OR**

The candidate description contains the template description as a substring (for non "" template descriptions) **OR**

The template description and name are both ""))

QUERY responses from server

The response format is a sequence of messages. For a successful query, e.g. that matched two resources:

```
{ "response" : "success" }
{ RESOURCE }
{ RESOURCE }
{ "resultSize" : 2 }
```

An example returned Resource is:

Server should NEVER reveal the owner!

```
{
    "name": "Unimelb website",
    "tags": [
        "web",
        "html"
    ],
    "description": "The main page for the University of Melbourne",
```

```
    "uri": "http:\/\/www.unimelb.edu.au",
    "channel": "",
    "owner": "",
    "ezserver": "aaron9010:3780"
}
```

Note that the **ezserver** field has been filled in by the server, to represent the server's hostname and port.

QUERY responses from server

The server will never reveal the owner of a resource in a response. If a resource has an owner then it will be replaced with the "*" character as in the following example:

```
{
    "name": "EZShare JAR",
    "tags": [
        "jar"
    ],
    "description": "The jar file for EZShare. Use with caution.",
    "uri": "file:\/\/\/\/home\/aaron\/EZShare\/ezshare.jar",
    "channel": "my_private_channel",
    "owner": "*",
    "ezserver": "aaron9010:3780"
}
```

This example also shows a resource that matched a channel, i.e. the user had specified the channel name in their query template.

QUERY responses from server

Other responses are related to standard errors.

If the resource template contained incorrect information that could not be recovered from:

```
{ "response" : "error",
  "errorMessage" : "invalid resourceTemplate"
}
```

If the resource or secret field was not given or not of the correct type:

```
{ "response" : "error",
  "errorMessage" : "missing resourceTemplate"
}
```

FETCH command

```
{
    "command": "FETCH",
    "resourceTemplate": {
        "name": "",
        "tags": [],
        "description": "",
        "uri": "file:\/\/\/\/home\/aaron\/EZShare\/ezshare.jar",
        "channel": "my_private_channel",
```

```
        "owner": "",
        "ezserver": null
    }
}
```

The role of the fetch command is to download the file resource from the server to the client.

Only the channel and URI fields in the template is relevant as it must be an exact match for the command to work.

Recall that, in a given channel, a given URI can only be present once, so that this command will only ever download a single file.

FETCH responses from server

A successful fetch will respond as follows:

```
{ "response" : "success" }
{ RESOURCE }
exact bytes of resource
{ "resultSize" : 1 }
```

The resource will have an additional field **resourceSize** that specifies the number of bytes (i.e. file size), e.g.:

```
{
    "name": "EZShare JAR",
    "tags": [
        "jar"
    ],
    "description": "The jar file for EZShare. Use with caution.",
    "uri": "file:\/\/\/\/home\/aaron\/EZShare\/ezshare.jar",
    "channel": "my_private_channel",
    "owner": "*",
    "ezserver": "aaron9010:3780",
    "resourceSize": 328515
}
```

The **resourceSize** field allows the client to read exactly the bytes of the file that follow.

FETCH responses from server

Other responses are related to standard errors.

If the resource template contained incorrect information that could not be recovered from:

```
{ "response" : "error",
  "errorMessage" : "invalid resourceTemplate"
}
```

If the resource template was not given or not of the correct type:

```
{ "response" : "error",
  "errorMessage" : "missing resourceTemplate"
}
```

EXCHANGE command

```
{
    "command": "EXCHANGE",
    "serverList": [
        {
            "hostname": "115.146.85.165",
            "port": 3780
        },
        {
            "hostname": "115.146.85.24",
            "port": 3780
        }
    ]
}
```

The purpose of the exchange command is to tell the server about a list of other servers.

The server is free to process any valid server record that it finds in the list and ignore others.

EXCHANGE responses from server

If the command succeeded:

```
{ "response" : "success" }
```

If a server record is found to be invalid:

```
{ "response" : "error",
  "errorMessage" : "missing resourceTemplate"
}
```

If the server list was missing or invalid:

```
{ "response" : "error",
  "errorMessage" : "missing or invalid server list"
}
```

Server Interactions

Each server maintains a list of Server Records, which are hostname:port strings. To begin, this list is empty.

Every X minutes (10 minutes by default, but configurable on the command line when the server is run), the server contacts a randomly selected server from the Server Records and initiates an EXCHANGE command with it. It provides the selected server with a copy of its entire Server Records list.

If the selected server is not reachable or a communication error occurs then the selected server is removed from the Server Records and no further action is taken in this round.

The receiving server processes the EXCHANGE command as explained earlier, essentially just adding the servers to its list.

QUERY relay

When a QUERY message is received with **relay** field set as **true** then the server sends a QUERY command to each of the servers in the Server Records list with the following change:

- *the owner and channel information in the original query are both set to "" in the forwarded query*
- **relay** field is set to **false**

Results returned from other servers are forwarded back to the original client on the same connection, aggregated with the results of the query processed locally. Therefore the response in the successful case is:

```
{ "response" : "success" }
{ RESOURCE }
{ RESOURCE }
...
{ "resultSize" : X }
```

where X is the number of hits, taking all of the results from other servers into account.

Connection Interval Limit

The server will ensure that the time between successive connections from any IP address will be no less than a limit (1 second by default but configurable on the command line).

An incomming request that violates this rule will be closed immediately with no response.

Client command line arguments

The client must work exactly with the following command line options:

```
-channel <arg>       channel
-debug               print debug information
-description <arg>   resource description
-exchange            exchange server list with server
-fetch               fetch resources from server
-host <arg>          server host, a domain name or IP address
-name <arg>          resource name
-owner <arg>         owner
-port <arg>          server port, an integer
-publish             publish resource on server
-query               query for resources from server
-remove              remove resource from server
-secret <arg>        secret
-servers <arg>       server list, host1:port1,host2:port2,...
-share               share resource on server
-tags <arg>          resource tags, tag1,tag2,tag3,...
-uri <arg>           resource URI
```

Example command lines

java -cp ezshare.jar EZShare.Client -query -channel my*private*channel -debug

java -cp ezshare.jar EZShare.Client -exchange -servers 115.146.85.165:3780,115.146.85.24:3780 -debug

java -cp ezshare.jar EZShare.Client -fetch -channel my*private*channel -uri file:///home/aaron/EZShare/ezshare.jar -debug

java -cp ezshare.jar EZShare.Client -share -uri file:///home/aaron/EZShare/ezshare.jar -name "EZShare JAR" -description "The jar file for EZShare. Use with caution." -tags jar -channel my*private*channel -owner aaron010 -secret 2os41f58vkd9e1q4ua6ov5emlv -debug

java -cp ezshare.jar EZShare.Client -publish -name "Unimelb website" -description "The main page for the University of Melbourne" -uri http://www.unimelb.edu.au -tags web,html -debug

java -cp ezshare.jar EZShare.Client -query

java -cp ezshare.jar EZShare.Client -remove -uri http://www.unimelb.edu.au

Server command line arguments

The server must work exactly with the following command line options:

```
-advertisedhostname <arg>        advertised hostname
-connectionintervallimit <arg>   connection interval limit in seconds
-exchangeinterval <arg>          exchange interval in seconds
-port <arg>                      server port, an integer
-secret <arg>                    secret
-debug                           print debug information
```

The default secret will be a large random string.

The default advertised host name will be the operating system supplied hostname.

The default exchange interval will be 10 minutes (600 seconds).

Example server output when just started

java -cp ezshare.jar EZShare.Server

20/03/2017 01:17:57.953 - [EZShare.Server.main] - [INFO] - Starting the EZShare Server

20/03/2017 01:17:57.979 - [EZShare.ServerControl.] - [INFO] - using secret: 5uv1ii7ec362me7hkch3s7l5c4

20/03/2017 01:17:57.981 - [EZShare.ServerControl.] - [INFO] - using advertised hostname: aaron9010

20/03/2017 01:17:57.984 - [EZShare.ServerIO.] - [INFO] - bound to port 3780

20/03/2017 01:17:57.986 - [EZShare.ServerExchanger.] - [INFO] - started

Debug command line option

The purpose of the debug option is that your system will print out every message sent or received, as in the following example for the client. So long as the words "SENT: ...msg..." and "RECEIVED: ...msg..." are present on a single line, it does not matter what else is on the same line (in this example the Java Logger is being used).

java -cp ezshare.jar EZShare.Client -publish -name "Unimelb website" -description "The main page for the University of Melbourne" -uri http://www.unimelb.edu.au -tags web,html -debug

20/03/2017 01:20:45.807 - [EZShare.Client.main] - [INFO] - setting debug on

20/03/2017 01:20:45.809 - [EZShare.Client.publishCommand] - [FINE] - publishing to localhost:3780

20/03/2017 01:20:45.865 - [EZShare.Client.sendMessage] - [FINE] - SENT: { "command" : "PUBLISH", "resource" : { "name" : "Unimelb website", "tags" : ["web", "html"], "description" : "The main page for the University of Melbourne", "uri" : "http://www.unimelb.edu.au", "channel" : "", "owner" : "", "ezserver" : null } }

20/03/2017 01:20:45.912 - [EZShare.Client.publishCommand] - [FINE] - RECEIVED: { "response" : "success" }

Technical aspects

- Requires Java 1.8 or above.
- Suggested to make use of the URI class to enforce URI rules.
- Make sure to use a JSON parser/formatter to generate correct JSON messages.
- Apache Commons CLI library is good for parsing command line options.
- Everyone should implement the same protocol, which means that clients and servers from different groups should interoperate fine.

Your Report

- Use 10pt font, double column, 1 inch margin all around.
- On the first page, clearly show your group's name and the names of all members in the group. Clearly show the login names with university emails as well. The members of the group MUST match the information entered into LMS.
- The report is aimed at addressing a number of questions discussed next. Have one section for in the report for each.
- Figures in the report, including examples of messages and protocol interaction, and any pseudo-code (that you may or may not use), are not counted as part of the word length guidelines.

Introduction

Write roughly 125 words to briefly describe in your own words:

- what the project was about

- what were the technical challenges that you faced building the system
- what outcomes did you achieve

Scalability

There are a number of aspects of the system that present a scalability challenge. In roughly 375 words:

- identify aspects of the system that present problems for scalability
    - ♦ be specific with why it is not scalable
- suggest revisions to the protocol that may overcome these problems

---

Concurrency

For a small system, with only a couple of servers and a few clients, concurrency issues are unlikely to arise. However consider a system with hundreds of servers and thousands of clients. There are some aspects of the system that may require further thought to ensure that concurrency issues are properly handled. Concurrency issues may include things that go technically wrong, but may also include things that do not work as well as expected.

In roughly 375 words describe:

- any concurrency issues that you identify, be specific with examples
- possible revisions to the protocol that may overcome these issues

---

Other Distributed System Challenges

failure models? connection failure? download resumption?

Choose a third distributed system challenge that relates to the system and write roughly 375 words explaining how it relates, how the system currently addresses the challenge (if it does at all), and how you might change the system to improve it with respect to the challenge.

Submission

You need to submit the following via LMS:

- Your report in PDF format *only*.
- Your ezshare.jar (that contains both Client and Server main classes as exemplified earlier).
- Your source files in a .ZIP or .TAR archive *only*.

Submissions will be due at the end of Week 8 via a group submission.