Time Stamped Anti-Entropy protocol

Author: Joan Manuel Marquès

Distributed Systems course

Spring 2013

1. Assignment Outline

The aim of this practical assignment is to implement and evaluate a weak-consistency protocol for data dissemination.

The project consist on:

- Implementing the Time Stamped Anti-Entropy (TSAE) protocol [1] into an application that stores cooking recipes in a set of replicated servers.
- Add a remove operation on the recipes application
- Evaluate how TSAE behaves under different conditions.

1.1 Groups

You are strongly advised to do the practical assignment in groups of 2 students, even though it is also possible to do it individually.

1.2 Evaluation

Up to phase 3 allows you to get a maximum mark of 8. (Maximum possible mark: 10).

Phase 4 allows you to get the maximum mark.

2. Overview

2.1 Application

The practical assignment implements the core mechanisms of a replicated application that stores cooking recipes. The application is formed by N servers, all of them having (eventually) a replica of all recipes.

Recipes have four fields:

- Title
- Recipe
- Author
- Timestamp

Two operations:

- Add recipe
- Remove recipe

Add and remove operations can be applied to any server.

Servers will converge to a common state (all servers having all recipes) by means of an epidemic algorithm, the Time Stamped Anti-Entropy (TSAE) protocol. This protocol guarantees that all servers will eventually have the same recipes, i.e. at a given time *t* not all server might have all recipes but in a finite unbounded time all serves will have all recipes issued (and not removed) before that time *t*.

2.2 TSAE protocol

Time Stamped Anti-Entropy (TSAE) [1] protocol is a weak-consistency protocol that provides

reliable and eventual delivery of issued operations.

To get an overview and main ideas about TSAE read from [2]:

1. Section *1.Introduction* (excluding subsection 1.1)
2. Section *2.2 Timestamped anti-entropy* (also interesting to read: *2.1 Kinds of consistency*)

Then, to get all details about the protocol, read *Chapter 5. Weak-consistency communication* from [1]. More precisely, read:

- *5.1.1 Data structures for timestamped anti-entropy*
- *5.1.2 The timestamped anti-entropy protocol*
- (If you implement purge) *5.3 Purging the message log* . Instead of using vector acks (loosely-synchronized clocks) as described in this section, you should use unsynchronized clocks, described in section *5.4.4 Anti-entropy with unsynchronized clocks* .

2.5 Your tasks

Phase 1: Implementation of log and timestamp vector data structures

Phase 2: Implementation of a reduced version of the application and TSAE protocol: only add operation; no purge of log

Phase 3: Extension of phase 2 to purge log with unsynchronized clocks

Phase 4: Evaluation of TSAE protocol and implementation of Remove recipe operation

3. Phases of the practical assignment

Phase 1: Implementation of log and timestamp vector data structures

Implement methods from `Log` and `TimestampVector` data structures.

In this phase only *add* operations are issued, therefore. Don't implement the functionality to purge the log.

Annex A includes details about the timestamps used in this practical assignment.

NOTE: be **careful with concurrent access to data structures**. Two actions issued by different threads may interleave. **Use** some **synchronization mechanism to avoid interferences**.

Phase 1.1 Your tasks

Implement Log and TimestampVector vector data structures.

Phase 2: Implementation of a reduced version of the application and TSAE protocol: only add operation; no purge of log

Implement the TSAE protocol described in section *5.1.2 The timestamped anti-entropy protocol* (from [1]) in the following classes (Package: `recipesService.tsaeDataStructures`):

- `TSAESessionOriginatorSide`: Originator protocol for TSAE (figure 5.7 without acks)
- `TSAESessionPartnerSide`: Partner's protocol for TSAE (figure 5.8 without acks)

`ServerData` class (package `recipesService`) contains `Server`'s data structures required

by the TSAE protocol (`log`, `summary`, `ack`) and the application (`recipes`).

- You can add any required method to allow `TSAESessionOriginatorSide` and `TSAESessionPartnerSide` manipulate these data structures.

Use the following methods to send and receive data between servers (package `communication`):

- `readObject`() from `ObjectInputStream_DS` class.
- `writeObject`() from `ObjectOutputStream_DS` class.

Use the following message classes for the communication between partners:

(package `recipesService.communication`)

- `MessageAErequest`: message sent to request an anti-entropy session.
- `MessageOperation`: message sent each time an operation is exchanged during an anti-entropy session.
- `MessageEndTSAE`: message sent to finish an anti-entropy session.

In this phase:

1. Only *add* operations (`addRecipe` method in `ServerData` class) are issued.
2. No purge of Log.

NOTE: be **careful with concurrent access to data structures**. Two actions issued by different threads may interleave. **Use** some **synchronization mechanism to avoid interferences**.

Phase 2.1 Your Tasks

Implement the protocol (without acks).

To test if your solution works properly use the provided test environment. Section 4 contains more details about it.

Phase 3: Extension of phase 2 to purge log with unsynchronized clocks

Extend previous phase adding all required logic to purge the log when using unsynchronized clocks:

- Implement purging method in `Log` class.
- Implement the necessary methods of `TimestampMatrix` according to section *5.4.4 Anti-entropy with unsynchronized clocks*.
- Extend the implementation of TSAE protocol of phase 2 to include ack exchange and to purge log (Section *5.3 Purging the message log*. Remember that you mush use unsynchronized clocks explained at section *5.4.4 Anti-entropy with unsynchronized clocks*)

In this phase only *add* operations are issued.

Phase 3.1 Your tasks

Implement the extensions above described and test them in the test environment.

Phase 4: Evaluation of TSAE protocol and implementation of Remove recipe operation

Phase 4.1 Extend application adding the *remove recipe* operation

Extend the cooking recipes application implementing the *remove recipe* operation:

1. Identify the problematics that introduce the *remove recipe* operation. Illustrate it with an example.
2. Design and implement a proposal to remove recipes and test it in the environment.

Modifications should be done in the following methods of `ServerData` class (package `recipesService`):

- `removeRecipe`
- any other required method added while implementing the TSAE protocol in phases 2.

Phase 4.2 Evaluation of TSAE protocol

Run the practical assignment under different conditions:

- scale: number of servers
- dynamicity: different degrees of connection and disconnection
- accelerate propagation: periodicity of anti-entropy sessions, start disseminating new data right after the data is generated, number of sessions with different partners that are done each time an anti-entropy session is scheduled.
- level of activity generation (add and remove operations)

Evaluate the impact of parameters on the behavior of TSAE. You should detail the experiments done and the obtained conclusions. Current implementation includes a basic modeling of parameters. You are encouraged to improve this modeling to get a more realistic one. Changes must be justified.

Phase 4.3 Your tasks

Implement the *remove* recipe *operation* and test it in the testing environment.

Do a rapport describing how parameters and running conditions influence on TSAE protocol performance.

4. Implementation and testing

4.1 Environment

Requires Java 7.

We recommend you to use eclipse as IDE. We will provide you an Eclipse project that contains the implementation of the cooking recipes application except the parts related to TSAE protocol.

All scripts for running local tests are prepared for Ubuntu-linux but other OS can be used. In that case, you will be responsible of adapting the scripts to your OS.

Use the script *start.sh* (included in `scripts` directory) to test your practical assignment locally.

4.2 Phase 1

To test `Log` and `TimestampVector` data structures run:

```
$  java -cp ../bin recipesService.Server --phase1
```

(run it from `scripts` folder)

Use the menu to introduce a recipe and check if `Log` and `TimestampVector` data structures are correct.

```
$  ./start.sh 20000 --phase1
```

(run it from `scripts` folder)

Executes `Log` and `TimestampVector` with a predefined set of users and operations and compares it with a `Log` and `TimestampVector` previously calculated.

4.3 Phases 2

To test your implementation locally in your computer use the shell script `start.sh` (`scripts` folder).

This script has many parameters. Some useful examples:

(arguments are explained only on its first appearance)

```
$  ./start.sh 20000 3 --menu –nopurge
```

Runs 3 Servers and a `TestServer` locally.

`$1`: listening port for the `TestServer`

`$2`: number of `Servers` to be instantiated

`--menu`: runs in menu mode. (without `--menu` argument, it will run in simulated mode, i.e. user activity (add recipes) and dynamism (connections and disconnections) will be automatically generated.

`--nopurge`: `Log` is not purged.

```
$  ./start.sh 20000 15 --logResults --nopurge --noremove
```

Runs 15 Servers and a `TestServer` locally.

User activity (add recipes) and dynamism (connections and disconnections) is automatically generated.

Results will be stored in a file named as the *groupId* from `config.properties` file (on current path).

`--logResults`: log results in the following two files:

- <*groupId*[2]>: log of all executions and the result for each of them.
- <*groupId*[3]>`.data`:

2 Name of the file will be the value of `groupId` property in `config.properties` file (`scripts` folder).

3 Value of `groupId` property in `config.properties` file (`scripts` folder) will be the first

- in the case that all solutions are equal, contains the final state of data structures.
- in the case that NOT all solutions are equal, contains the first two found solutions that were different.

`--nopurge: Log` is not purged.

`--noremove`: no remove operations are issued.

```
$ ./start.sh 20000 15 --logResults -path ../results --nopurge --noremove
```

`-path <path>`: result files are created in `../results` folder. If no `-path` is specified (as in previous example) files will be stored in current folder.

We recommend you to **start** testing your application using the `--menu` option, which will allow you to test the TSAE protocol and data structures at your pace.

Then execute your implementation locally without the `--menu` parameter.

4.4 Phases 3

```
$ ./start.sh 20000 3 --logResults -path ../results --menu
```

Runs 3 Servers and a `TestServer` locally (and result files are created in `../results` folder).

`Log` is purged.

`Menu` mode.

```
$ ./start.sh 20000 15 --logResults -path ../results --noremove
```

Runs 15 Servers and a `TestServer` locally (and result files are created in `../results` folder).

`Log` is purged.

Simulated activity and dynamism mode.

No remove operations.

4.5 Phases 4

Use the script *start.sh* explained in previous phases.

In *phase 4.1* (extend application adding remove recipe operation), run *start.sh* script to test your implementation.

```
$ ./start.sh 20000 15 --logResults -path ../results
```

Runs 15 Servers and a `TestServer` locally (and result files are created in `../results` folder).

`Log` is purged.

---

part of the file name. i.e. file name: <value of groupId property>`.data`.

Simulated activity and dynamism mode.

Remove operations are issued.

In *phase 4.2* (evaluation of TSAE protocol), modify parameters of `config.properties` file (`scripts` folder) to evaluate TSAE protocol under different conditions and environments.

- Use *runN.sh* script to run N times *start.sh* script. Example:

  ```
  $ ./runN.sh 10 20000 15 --logResults -path ../results
  ```

  runs 10 times *start.sh* script with the following parameters:

  ```
  $ ./start.sh 20000 15 --logResults -path ../results
  ```

- Modify `activitySimulation` class only in case that in phase 4 you want to better understand or modify the modeling of activity and dynamicity (package `recipesService.activitySimulation`).

5. Things to deliver

You should send a zip file. **Name** the **file** according to the following convention:

Year-groupId-FamilyName1_Name1-FamilyName2_Name2.zip

This file should **include**:

- For Phase 4: required docs (name file *phase4.pdf*)
- A (short) report detailing and arguing all decisions taken. A proposal of report template is included in the practical assignment distribution (*/doc* folder).

  In addition, you should detail the portions of your source code you consider are most important, an explanation about how it works and the tests you used to validate your assignment.

  Finally, scripts and a detailed example of how to run your practical assignment.

- Source code: your eclipse project and the portion of source code you have implemented.

The zip file should have a single directory named like the zip file with the following structure (use same structure and names):

Subdirectory	Content
`/doc`	Report in pdf Other docs (phase 4)
`/src`	Source code. Eclipse project with your implementation.
`/bin`	All required files to execute your practical assignment. Explained carefully in the report how to execute it.
`/sol`	Your solution. Only include the code that you have done, i.e. the classes you implemented or modified. i.e. the files from `/src` that you have implemented, without subdirectories. Only the .java.

Annex A. Source code and documentation

`Papers` folder contains referenced paper and thesis that explain TSAE protocol.

`TSAE` folder contains all packages and classes required to the practical assignment.

Important: **do not implement new classes**. **Modify only the classes indicated in each phase**.

(except if you modify the basic modeling of parameters in phase 4)

Classes that you should modify:

1. package `recipesService.tsaeDataStructures`:
   - `Log`: class that logs operations.
   - `TimestampVector`: class to maintain the summary.
   - `TimestampMatrix`: class to maintain the acknowledgment matrix of timestamps
2. package `recipesService.tsaeDataStructures`:
   - `TSAESessionOriginatorSide`: Originator protocol for TSAE.
   - `TSAESessionPartnerSide`: Partner's protocol for TSAE.
3. package `recipesService`:
   - `ServerData`: contains `Server`'s data structures required by the TSAE protocol (`log`, `summary`, `ack`) and the application (`recipes`). You can add any required method to allow `TSAESessionOriginatorSide` and `TSAESessionPartnerSide` manipulate these data structures.
     - `addRecipe` method: adds a new recipe.
     - `removeRecipe` method: removes a recipe.

Classes that you should use but NOT modify:

4. package `recipesService.tsaeDataStructures`:
   - `Timestamp`: a timestamp. A timestamp allows the ordering of operations issued from same host. It is a tuple <hostId, sequenceNumber>. Sequence number is a number that grows monotonically. The first valid timestamp issued by a host will have an initial value of 0. A negative sequence number means that the host hasn't issued yet any operation. Timestamps can not be used to order operations issued in different hosts. Next timestamp is obtained calling the method `nextTimestamp()` from class `ServerData` (package `recipesService`).
5. package `recipesService.data`:
   - `AddOperation`: an add operation (operations are logged in the `Log` and exchanged with other partners).
   - `RemoveOperation`: a remove operation (operations are logged in the `Log` and exchanged with other partners).
   - `Recipe`: a recipe.
   - `Recipes`: class that contains all recipes.
6. package `recipesService.communication`)

- `MessageAErequest`: message sent to request an anti-entropy session.
- `MessageOperation`: message sent for each exchanged operation during an anti-entropy session.
- `MessageEndTSAE`: message sent to finish an anti-entropy session.

7. package `communication`:
   - `ObjectInputStream_DS`: class that implements a modification of the `ObjectInputStream` to simulate failures. Use the method `readObject`().
   - `ObjectOutputStream_DS`: class that implements a modification of the `ObjectOutputStream` to simulate failures. Use the method `writeObject`().
8. package `recipesService.activitySimulation:` Only in case that in phase 4 you want to better understand or modify the modeling of activity and dynamicity.

References

[1] **Richard A. Golding** (1992, December). Weak-consistency group communication and membership. Ph.D. thesis, published as technical report UCSC-CRL-92-52. Computer and Information Sciences Board, University of California. Santa Cruz. **(chapter 5)**

[2] **R. Golding; D. D. E. Long**. The performance of weak-consistency replication protocols, UCSC Technical Report UCSC-CRL-92-30, July 1992.