Time Stamped Anti-Entropy protocol

Authors: Joan Manuel Marquès, Antonio González, Raúl de la Cruz and José Quiroga

Distributed Systems course

Spring 2021

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.

Time Stamped Anti-Entropy (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* .

Groups

Phase 1: should be done **individually**.

You are strongly advised to do the **phases 2 to 4 in groups of 2 students**, even though it is also possible to do it individually.

To Deliver

Two deliverables (more details in each phase):

1. phase 1 (theoretical exercises and practice)
2. phases 2 to 4 or second theoretical exercise

Deadlines are in the course schedule.

D1. First Deliverable

The work of this delivery must be done **individually**.

1. Phase 1: Theoretical exercise of TSAE protocol

1.1 TSAE protocol exercise (no purged log)

Exercise 1

Let's suppose that there are 3 hosts (A, B, C) that use TSAE protocol to exchange operations. At the initial time (t0) all hosts have the same state and their logs and summary vectors are as follow:

Summary A= A3, B3, C2 Log A= A1, A2, A3, B1, B2, B3, C1, C2

Summary B= A1, B4, C2 Log B= A1, B1, B2, B3, B4, C1, C2

Summary C= A2, B3, C4 Log C= A1, A2, B1, B2, B3, C1, C2, C3, C4

A1 stands for first operation from host A, B2 stands for the second operation from host B, and so on.

Let's assume that each anti-entropy session starts and ends at the same instant.

1.1.1 For the following sequence:

Time	Operation
1	Host A executes operation A4
2	Host B does an anti-entropy session with host C
3	Host C executes operations C5
4	Host A does an anti-entropy session with host C
5	Host A executes operation A5
6	Host B executes operations B5
7	Host A does an anti-entropy session with host B
8	Host C executes operations C6
9	Host B executes operation B6, B7, B8
10	Host C executes operations C7, C8
11	Host A executes operations A6
12	Host B does an anti-entropy session with host C
13	Host C does an anti-entropy session with host A
14	Host A executes operation A7

We want you to provide us with:

1. The data structures and operations exchanged during the anti-entropy sessions
2. The data structures (log and summary) at each host after each anti-entropy session
3. Indicate if final state is consistent, i.e. all hosts have received the same operations. In case it is not consistent, indicate which sessions should be done to reach a consistent state.

(Note: log is not purged)

T1:

Summary A= A4, B3, C2 Log A= A1, A2, A3, A4, B1, B2, B3, C1, C2
Summary B= A1, B4, C2 Log B= A1, B1, B2, B3, B4. C1, C2
Summary C= A2, B3, C4 Log C= A1, A2, B1, B2, B3, C1, C2, C3, C4

T2:

Host B sends to host C: B4

Host C sends to host B: A2, C3, C4

Summary A= A4, B3, C2 Log A= A1, A2, A3, A4, B1, B2, B3, C1, C2
Summary B= A2, B4, C4 Log B= A1, A2, B1, B2, B3, B4, C1, C2, C3, C4
Summary C= A2, B4, C4 Log C= A1, A2, B1, B2, B3, B4, C1, C2, C3, C4

T3:

Summary A= A4, B3, C2 Log A= A1, A2, A3, A4, B1, B2, B3, C1, C2
Summary B= A2, B4, C4 Log B= A1, A2, B1, B2, B3, B4, C1, C2, C3, C4
Summary C= A2, B4, C5 Log C= A1, A2, B1, B2, B3, B4, C1, C2, C3, C4, C5

T4:

Host A sends to host C: A3, A4

Host C sends to host A: B4, C3, C4, C5

Summary A= A4, B4, C5 Log A= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5
Summary B= A2, B4, C4 Log B= A1, A2, B1, B2, B3, B4, C1, C2, C3, C4
Summary C= A2, B4, C5 Log C= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5

T5:

Summary A= A5, B4, C5 Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, C1, C2, C3, C4, C5
Summary B= A2, B4, C4 Log B= A1, A2, B1, B2, B3, B4, C1, C2, C3, C4
Summary C= A2, B4, C5 Log C= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5

T6:

Summary A= A5, B4, C5 Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, C1, C2, C3, C4, C5
Summary B= A2, B5, C4 Log B= A1, A2, B1, B2, B3, B4, B5, C1, C2, C3, C4
Summary C= A2, B4, C5 Log C= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5

T7:

Host A sends to host B: A3. A4. A5, C5

Host B sends to host A: B5

Summary A= A5, B5, C5 Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5
Summary B= A5, B5, C5 Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5
Summary C= A2, B4, C5 Log C= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5

T8:

Summary A= A5, B5, C5 Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5
Summary B= A5, B5, C5 Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5
Summary C= A2, B4, C6 Log C= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5, C6

T9:

Summary A= A5, B5, C5 Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5
Summary B= A5, B8, C5 Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5
Summary C= A2, B4, C6 Log C= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5, C6

T10:

Summary A= A5, B5, C5 Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5
Summary B= A5, B8, C5 Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5
Summary C= A2, B4, C8 Log C= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5, C6, C7, C8

T11:

Summary A= A6, B5, C5 Log A= A1, A2, A3, A4, A5, A6, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5
Summary B= A5, B8, C5 Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5
Summary C= A2, B4, C8 Log C= A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, C5, C6, C7, C8

T12:

Host B sends to host C: A3, A4, A5, B5, B6, B7, B8

Host C sends to host B: C6, C7, C8

Summary A= A6, B5, C5 Log A= A1, A2, A3, A4, A5, A6, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5
Summary B= A5, B8, C8 Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8
Summary C= A5, B8, C8 Log C= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8

T13:

Host C sends to host A: B6, B7, B8, C6, C7, C8

Host A sends to host C: A6

Summary A= A6, B8, C8 Log A= A1, A2, A3, A4, A5, A6, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8
Summary B= A5, B8, C8 Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8
Summary C= A6, B8, C8 Log C= A1, A2, A3, A4, A5, A6, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8

T14:

Summary A= A7, B8, C8 Log A= A1, A2, A3, A4, A5, A6, A7, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8
Summary B= A5, B8, C8 Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8
Summary C= A6, B8, C8 Log C= A1, A2, A3, A4, A5, A6, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8

*Hosts B and C have pending operations to receive from host A. The final state is **not** consistent.*

1.1.2 For the following sequence:

Time	Operation
1	Host A executes operation A4
2	Host A does an anti-entropy session with host C
3	Host B executes operation B5, B6
4	Host A does an anti-entropy session with host B
5	Host C executes operation C5
6	Host A executes operation A5
7	Host C does an anti-entropy session with host A
8	Host A does an anti-entropy session with host B

We want you to provide us with:

1. The data structures and operations exchanged during the anti-entropy sessions
2. The data structures (log and summary) at each host after each anti-entropy session
3. Indicate if final state is consistent, i.e. all hosts have received the same operations. In case it is not consistent, indicate which sessions should be done to reach a consistent state.

(Note: log is not purged)

T1:
Summary A= A4, B3, C2 Log A= A1, A2, A3, A4, B1, B2, B3, C1, C2
Summary B= A1, B4, C2 Log B= A1, B1, B2, B3, B4. C1, C2
Summary C= A2, B3, C4 Log C= A1, A2, B1, B2, B3, C1, C2, C3, C4

T2:

Host A sends to host C: A3, A4

Host C sends to host A: C3, C4

Summary A= A4, B3, C4 Log A= A1, A2, A3, A4, B1, B2, B3, C1, C2, C3, C4
Summary B= A1, B4, C2 Log B= A1, B1, B2, B3, B4. C1, C2
Summary C= A4, B3, C4 Log C= A1, A2, A3, A4, B1, B2, B3, C1, C2, C3, C4

T3:

Summary A= A4, B3, C4 Log A= A1, A2, A3, A4, B1, B2, B3, C1, C2, C3, C4
Summary B= A1, B6, C2 Log B= A1, B1, B2, B3, B4. B5, B6, C1, C2
Summary C= A4, B3, C4 Log C= A1, A2, A3, A4, B1, B2, B3, C1, C2, C3, C4

T4:

Host A sends to host B: A2, A3, A4, C3, C4

Host B sends to host A: B4, B5, B6

Summary A= A4, B6, C4 Log A= A1, A2, A3, A4, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4
Summary B= A4, B6, C4 Log B= A1, A2, A3, A4, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4
Summary C= A4, B3, C4 Log C= A1, A2, A3, A4, B1, B2, B3, C1, C2, C3, C4

T5:

Summary A= A4, B6, C4 Log A= A1, A2, A3, A4, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4
Summary B= A4, B6, C4 Log B= A1, A2, A3, A4, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4
Summary C= A4, B3, C5 Log C= A1, A2, A3, A4, B1, B2, B3, C1, C2, C3, C4, C5

T6:

Summary A= A5, B6, C4	Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4
Summary B= A4, B6, C4	Log B= A1, A2, A3, A4, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4
Summary C= A4, B3, C5	Log C= A1, A2, A3, A4, B1, B2, B3, C1, C2, C3, C4, C5

T7:

Host C sends to host A: C5

Host A sends to host C: A5, B4, B5, B6

Summary A= A5, B6, C5	Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4, C5
Summary B= A4, B6, C4	Log B= A1, A2, A3, A4, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4
Summary C= A5, B6, C5	Log C= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4, C5

T8:

Host A sends to host B: A5. C5

Host B sends to host A: -

Summary A= A5, B6, C5	Log A= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4, C5
Summary B= A5, B6, C5	Log B= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4, C5
Summary C= A5, B6, C5	Log C= A1, A2, A3, A4, A5, B1, B2, B3, B4, B5, B6, C1, C2, C3, C4, C5

All hosts have received the same operations. The final state is consistent.

1.2 TSAE protocol exercise (purged log)

Imagine a situation with 5 hosts with unsynchronized clocks and log purging where Hosts A and B have the following AckSummaries:

AckSummary A

A	A 5	B 2	C 1	D 2	E 3
B	A 4	B 3	C 4	D 2	E 4
C	A 2	B 2	C 5	D 2	E 2
D	A 3	B 3	C 4	D 3	E 4
E	A 4	B 3	C 1	D 1	E 4

AckSummary B

A	A 4	B 2	C 3	D 4	E 5
B	A 2	B 4	C 3	D 2	E 5
C	A 3	B 2	C 3	D 3	E 5
D	A 3	B 1	C 3	D 4	E 5
E	A 4	B 1	C 3	D 4	E 5

a) Which is the content of Log in host A?

A3, A4, A5, B3, C2, C3, C4, C5, D2, D3, E3, E4

b) Which is the content of Log in host B?

A3, A4, B2, B3, B4, D3, D4

A and B do an anti-entropy session. During the session both know who is each other (this is different from the algorithm in the Golding thesis).

c) Which operations are exchanged during the anti-entropy session?

Host A sends to host B: A5, C4, C5,

Host B sends to host A: B4, D4, E5

d) Which AckSummary and log have each host after ending the session?

AckSummary A, B

A	A5	B4	C5	D4	E5
B	A5	B4	C5	D4	E5
C	A3	B2	C5	D3	E5
D	A3	B3	C4	D4	E5
E	A4	B3	C3	D4	E5

Log D, E: A4, A5, B3, B4, C4, C5, D4

1.3 TSAE protocol exercise

Question 1: In TSAE, why is it needed to purge the log periodically? Explain with your own words the question.

A possible answer to this question is: The maintenance task is necessary to avoid the uncontrolled growth of the log files, furthermore, the correct purging must be done without interfering with the message propagation.

Question 2: Describe what are the main fields of a TSAE data structure.

The main fields are the tuple of the host ID and the clock, the host ID can represent either the host name or the unique host ID. The clock is the current clock of the host. The combination of these fields can give a unique time stamp for a host. Timestamps can be organized into timestamp vectors, which is a set of timestamps, indexed by their host IDs. The last structure is the message log, what contains messages that have been received by a principal. To conclude, TSAE needs 3 basic structures: the timestamp, the timestamp vector and the message log.

2. Phase 1: Implementation and testing of Log and TimestampVector data structures

Phase 1 will consist on implementing the methods from `Log` and `TimestampVector` data structures.

In this phase only *add* operations are issued. 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**.

Implement `Log` and `TimestampVector` data structures according to TSAE protocol described in section *Time Stamped Anti-Entropy (TSAE) protocol*.

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.

2.1. Test locally

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

- `java -cp ../bin:../lib/* recipes_service.Server --phase1`

  (run it from `scripts` folder)

  Introduce a recipe and check if `Log` and `TimestampVector` data structures are correct.

- `./start.sh 20004 --phase1`

  (run it from `scripts` folder)

  $1: listening port of `Phase1TestServer`, the sever that tests the correctness of your solution. In case that port `20004` is already in use by another application you can change it to any other unused port.

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

2.2. Formal evaluation of phase 1

It is compulsory to **use DSLab to assess the phase 1** of the practical assignment. Therefore, once your application runs in local, upload `Log` and `TimestampVector` classes into DSLab (https://sd.uoc.edu/dslab) and assess them:

1. Create a *Project* and upload `Log` and `TimestampVector` classes into this project.
2. Create an *Experiment* that executes the project created in step 1.

3. Check the result of the execution of step 2.

2.3. Things to deliver

A **file** according to the following convention:

Deliverable1-Year-FamilyName1_Name1.zip

This file should **include**:

- solution of the theoretical exercise.
- phase1.pdf: a (short) report detailing and explaining 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 significant.
- Source code: `Log` and `TimestampVector` classes.

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`	Solution theoretical exercise Report in pdf
`/src`	`Log` and `TimestampVector` classes.

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 `recipes_service.tsae.datastructures`:
   - `Log`: class that logs operations.
   - `TimestampVector`: class to maintain the summary.
   - `TimestampMatrix`: class to maintain the acknowledgment matrix of timestamps
2. package `recipes_service.tsae.sessions`:
   - `TSAESessionOriginatorSide`: Originator protocol for TSAE.
   - `TSAESessionPartnerSide`: Partner's protocol for TSAE.
3. package `recipes_service`:
   - `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 `recipes_service.tsae.data_structures`:
   - `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 `recipes_service`).
5. package `recipes_service.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 `recipes_service.communication`)

- `MessageAErequest`: message sent to request an anti-entropy session.
- `MessageOperation`: message sent for each exchanged operationTimes New Roman 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 `recipes_service.activitysimulation:` Only in case that in phase 4 you want to better understand or modify the modeling of activity and dynamicity.

Annex B. Activity simulation and dynamicity

Simulation of communication failures

`ActivitySimulation` class (`package recipes_service.activitysimulation`) decides when to simulate the disconnection (or network failure) of a host and when to reconnect it.

To simulate communication failures, a disconnects host abruptly closes all its Input and Output streams, what results in an exception in the two partners that are using the stream.

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.