DADSTORM

Márcio Santos, Diogo Ferreira, Francisco Santos
76338, 79018, 79719
Group 24
Instituto Superior Técnico
Av. Rovisco Pais, 1, 1049-001 Lisboa, Portugal
Paolo.Ienne@tecnico.ulisboa.pt, jikj, franciscopolaco@tecnico.ulisboa.pt

Abstract

Stream processing as become popular, so it is normal that there is a lot of technologies out there concerning it. The scope of our work is to design and implement a simplified distributed fault-tolerant stream processing system, for education purposes only. In our system, the programmer can, very easily, write a configuration file and some custom functions, to give his program the correct semantic while running on top of our platform.

1. Introduction

Stream processing consist in given a sequence of data (tuples in our case) and apply a set of operations to it. When we take this to a distributed level other problems, like crashes, can appear, so it is required to ensure the correctness of the computation by using algorithms that apply fault-tolerant semantics.

Some concepts that are needed to understand our work are explain in the subsections. To facilitate the apprehension we will apply a metaphor, in which we will pretend that we have a factory, where we receive a raw material, work on that and produce something new.

1.1 Replica

A **Replica** or as we called them in the **Design** section, a **Slave**, is the most basic unit of our system. It is manly responsible to execute an operations over a set of tuples. They also are used to ensure replication and fault-tolerant properties. In our metaphor, this concept are mapped as assembly lines, where every line does the same, and if some stops, the others assure that the work is done.

1.2 Operator

An **Operator** is a set of **Replicas** that do the same operation. This set is responsible to tolerate faults within it self. In our metaphor, we mapped this concept, as the factory it self, where it receives tuples, process them and produce new tuples.

2. Design

One of the key aspects of every solution is its design, especially in distributed systems. So, having a good design was a top priority for us. Our design provides a highly flexible and scalable environment, which adapts to every need.

First, we will discuss the components of the system, their roles and modules. Then we will dive into the relations between them. Present the must relevant architecture and finish with a brief overview.

2.1. Components

Our system is divided in four main components, which are the Puppet Master, Process Creation Service, Slave and Common Types.

2.1.1 Roles

Puppet Master reads the configuration file, processes it and sends the needed commands to the Process Creation Service of each machine. After that, it will only collect logs from all replicas and serve as a command prompt. It basically is a graphical interface where we can easily bootstrap, monitor and manipulate the system.

Process Creation Service (PCS) is responsible for ordering the creation of slaves, i.e. the replicas and for feeding the first tuples into the first operator. In some sense, the process creation service is just a simple parser.

Slave is in charge of the actual processing of the tuples. The processing happens in three, totally decoupled phases: importing, processing and routing.

Common Types is a shared library between the different components. Which, defines the interfaces and specialized objects for data transfers.

2.1.2 Modules

In this section we will approach which units of implementation - modules - are the components of the system composed of.

The Puppet Master, has the following modules:

Logger: logs the events, received from the slaves, by writing it to the screen and to a file.

Operator & Proxies: the operator is a composition of proxies. Each proxy is a remote object of a slave, which we use to send commands, asynchronously, after their creation.

Configuration File Processor: reads the configuration file and parses it into chunks of information which we send to the PCS.

The Slave, has the following modules:

Importer. source of the tuples. We can distinguish three importing types: Input, File and Operator imports. The difference between Input and File Import is that the first one is only used for the first operator of the chain, since its data comes from the PCS. This decision came because of the random policy for importing, we wanted to be sure that all the tuples were imported. The Operator import represents an operator that receives its input from a upstream operator.

Processor. given a tuple applies the domain logic. There are two types of processors: the stateless and stateful. The stateless, are composed by the Dup, Filter and Custom. The stateful, have the Uniq and Count. Its logic is described in the project description.

Routing. routes the processed tuple. There are four different types of routing: Primary, Random, Hashing and Output. The first three obey the domain logic. The last one came from the necessity of outputting the resulting tuples, in the last operator, into a file.

Note that the PCS and Common Types aren't composed of modules, due to their simplicity.

2.2. Relations

The flow begins with the Puppet Master when he loads the configuration file. After loading it, the Puppet Master orders the PCS to create the operators and their respective replicas. When the PCS finishes, the Puppet Master starts sending commands to the Slaves. The Slaves, according to the required characteristics of the system do their work in a pipe-and-filter manner, notifying the Puppet Master Logger when they do something.

2.3. Architecture

In the following section we pretend to expose the qualities of our solution. Note that the diagrams that will be presented are not complete, attribute or operation wise to simplify.

As we talked the importing, processing and routing are totally decoupled. In order to achieve that, we used a **Abstract Factory Pattern**, which allows us to create different specialized factories to fit our needs. So to create a replica we just need to "ask" each factory for a concrete type, forming any combination of Slaves that we would like. This approach gives us **plenty of flexibility**, as shown in **fig. 1**. So imagine that you want to create a Slave that imports from another operator, processes with uniq and routes for a file. You just have to tell the importing factory to give you a OpImport, the processing factory to give you a Uniq and finally the routing factory to give a Output.

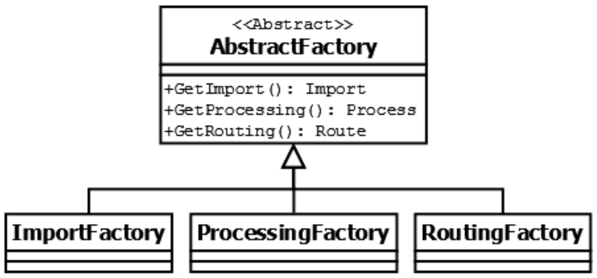

Figure 1: The different factories used to create the replicas.

Another important aspect of our design is how the Slave operates under different states. As we know, there are, currently, on the project the Freeze and Unfrozen states. Depending on the state, the Slave should behave differently. To achieve such aim dynamically and in a scalable fashion, we used the **State Pattern fig. 6**. With this style we do not need to worry about the logic of the operations, since its behavior is decided on runtime, according to whatever the Puppet Master "throws" at us.

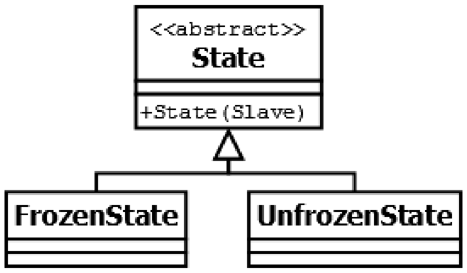

Figure 2: The different states.

It is worth mentioning that in the Puppet Master the Logger being a separated module, unburdens the Puppet Master of that ”tedious job” allowing it to process the commands given to him, i.e. does not freeze the UI, which allows to achieve the interactive model - command line.

2.4. Overview

2.5. Nop

Wherever Times is specified, Times Roman may also be used. If neither is available on your word processor, please use the font closest in appearance to Times that you have access to.

MAIN TITLE. Center the title 1-3/8 inches (3.49 cm) from the top edge of the first page. The title should be in Times 14-point, boldface type. Capitalize the first letter of nouns, pronouns, verbs, adjectives, and adverbs; do not capitalize articles, coordinate conjunctions, or prepositions (unless the title begins with such a word). Leave two blank lines after the title.

AUTHOR NAME(s) and AFFILIATION(s) are to be centered beneath the title and printed in Times 12-point, non-boldface type. This information is to be followed by two blank lines.

The ABSTRACT and MAIN TEXT are to be in a two-column format.

MAIN TEXT. Type main text in 10-point Times, single-spaced. Do NOT use double-spacing. All paragraphs should be indented 1 pica (approx. 1/6 inch or 0.422 cm). Make sure your text is fully justified—that is, flush left and flush right. Please do not place any additional blank lines between paragraphs. Figure and table captions should be 10-point Helvetica boldface type as in

Figure 3: Example of caption.

Long captions should be set as in

Figure 4: Example of long caption requiring more than one line. It is not typed centered but aligned on both sides and indented with an additional margin on both sides of 1 pica.

Callouts should be 9-point Helvetica, non-boldface type. Initially capitalize only the first word of section titles and first-, second-, and third-order headings.

FIRST-ORDER HEADINGS. (For example, **1. Introduction**) should be Times 12-point boldface, initially capitalized, flush left, with one blank line before, and one blank line after.

SECOND-ORDER HEADINGS. (For example, **1.1. Database elements**) should be Times 11-point boldface, initially capitalized, flush left, with one blank line before, and one after. If you require a third-order heading (we discourage it), use 10-point Times, boldface, initially capitalized, flush left, preceded by one blank line, followed by a period and your text on the same line..

2.6. Footnotes

Please use footnotes sparingly[1] and place them at the bottom of the column on the page on which they are referenced. Use Times 8-point type, single-spaced.

2.7. References

List and number all bibliographical references in 9-point Times, single-spaced, at the end of your paper. When referenced in the text, enclose the citation number in square brackets, for example [1]. Where appropriate, include the name(s) of editors of referenced books.

2.8. Illustrations, graphs, and photographs

All graphics should be centered. Your artwork must be in place in the article (preferably printed as part of the text rather than pasted up). If you are using photographs and are able to have halftones made at a print shop, use a 100- or 110-line screen. If you must use plain photos, they must be pasted onto your manuscript. Use rubber cement to affix the images in place. Black and white, clear, glossy-finish photos are preferable to color. Supply the best quality photographs and illustrations possible. Penciled lines and very fine lines do not reproduce well. Remember, the quality of the book cannot be better than the originals provided. Do NOT use tape on your pages!

2.9. Color

The use of color on interior pages (that is, pages other than the cover) is prohibitively expensive. We publish interior pages in color only when it is specifically requested and budgeted for by the conference organizers. DO NOT SUBMIT COLOR IMAGES IN YOUR PAPERS UNLESS SPECIFICALLY INSTRUCTED TO DO SO.

2.10. Symbols

If your word processor or typewriter cannot produce Greek letters, mathematical symbols, or other graphical elements, please use pressure-sensitive (self-adhesive) rub-on symbols or letters (available in most stationery stores, art stores, or graphics shops).

[1]Or, better still, try to avoid footnotes altogether. To help your readers, avoid using footnotes altogether and include necessary peripheral observations in the text (within parentheses, if you prefer, as in this sentence).

3 Evaluation

We setup some configuration files in order to test our system. Each test has a specific focus, for instance we evaluate if a stream reach the end even when a replica crashes.

All the following tests were ran in the three possible semantics, however we focus more on the at-least-once semantic, since is not trivial to implement and on the exactly-once semantic, because is the most difficult to implement. Note that at-most-once semantic does not even assure that a tuple will reach the end, or that the operators have their state synchronized, so it will not be shown in detail like the others.

Also, before reaching to the critical point, we assured that all tested had tuples that would make the operator do something instead of a trivial case, where it would do nothing.

3.1 Environment Setup

Since we want people to be able to reproduce these tests, we present our environment setup. Every test was ran locally, since we didn't had the resources to have various Windows machines in the same network.

Our environment was:

- Windows 10 Pro x64
- Intel Core i7-6700HQ @ 2.60GHz
- 16 GB DDR4 RAM
- Microsoft Visual Studio Enterprise 2015
- Microsoft .NET Framework 4.5.2

3.2 State Synchronization

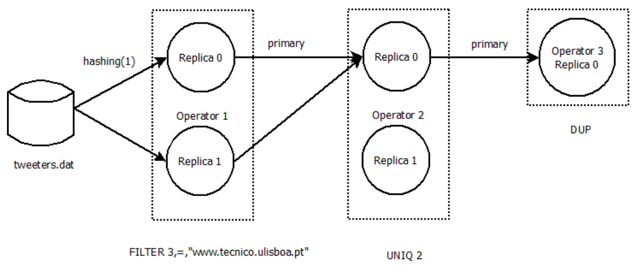

Figure 5: State Synchronization Test

In this test, we hoped to verify if the state was being synchronized in the second operator since we froze replica 0 and some 7 seconds after we unfreeze it.

This test does not have any semantic per se, however the objective was to check if there was any tuple in the output with same user.

3.2.1 at-least-once semantic

All the tuples reach the end, sometimes in duplicated. This happens because we send the tuples to replica 0 of operator 2, but it is frozen. Since operator 1 does not receive an *ACK*, it resends to other replica. Later, when replica 0 is unfrozen, it will process everything, which will result in duplicated tuples.

3.2.2 exactly-once semantic

Here the output will be slower, as it is expected, since siblings are asked if a tuple was seen. Nevertheless, even with the frozen period outputs what should output if no failure had occur.

3.3 Example in the project description

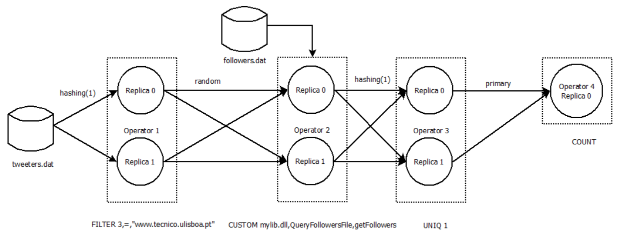

Figure 6: Crash Test

In this test, we wanted to check if we were able to keep the semantics with more operators and by crashing a replica. This test should output the number of user that had contact with a post related with *"www.tecnico.ulisboa.pt"*.

3.3.1 at-least-once semantic

Again all the tuples reach the end, sometimes in duplicated.

3.3.2 exactly-once semantic

Every tuple reach the end, as if crashes or freezes had not occur.

4 Conclusions

Somos um gajos fixes e sexys :D

References

[1] I. M. Author. Some related article I wrote. *Some Fine Journal*, 99(7):1–100, January 1999.

[2] A. N. Expert. *A Book He Wrote*. His Publisher, Erewhon, NC, 1999.