Distributed Local Sequence Alignment

DISTRIBUTED COMPUTING SYSTEMS (X_400130)
LARGE LAB EXERCISE

Tiziano De Matteis

V1.0

1 Learning Objectives

1. Design and implement a distributed version of a widespread sequence alignment algorithm.
2. Design and implement experiments for the distributed system.
3. Evaluate the implementation using realistic workloads.

This exercise combines elements of Chapter 4 (communication), Chapter 6 (synchronization), Chapter 7 (consistency and replication), and Chapter 8 (fault tolerance) of the course text book (Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems: Principles and Paradigms, third edition, Prentice Hall, 2017).

2 Assignment Description

Sequence alignment is a fundamental problem in bioinformatics and computational biology. Given two strings D (the database) and S (the pattern), the alignment problem consists in finding an arrangement of the two strings to identify regions of similarity. Usually, a similarity in a fragment of DNA (or RNA or protein) sequences means a structural or functional similarity. Therefore, sequence alignment is used to understand the evolutionary relationships, structural features, and functional properties of such sequences.

We can distinguish between two main possible types of alignment:

- *global alignment*, where we are interested in finding an alignment that spans the entire length of the two strings;
- *local alignment*, that is an alignment between a substring of D and a substring of S.

In this project, we focus on *local alignments*. In particular, you will have to design and implement a distributed version of the Smith-Waterman sequence alignment algorithm [Smith and Waterman(1981)] to improve its performance and scalability for handling large biological sequence datasets.

2.1 Requirements

Mandatory Requirements

- ☐ Design and implement a distributed version of the Smith-Waterman algorithm capable of scaling across multiple machines. The application must return the best-found alignment.

- ☐ Design and implement experiments testing the system's scalability by aligning sequences of varying lengths and sizes, up to large genomic datasets.
- ☐ Analyze your experiment data and report your insights.
- ☐ All your experiments must be reproducible without significant effort by other users.

Bonus Features – You Will Receive Bonus Points for at Most 4 Implemented Features

- ☐ By default, the Smith-Waterman algorithm returns (one of) the alignment(s) obtaining the best score. Modify your implementation to return the top-k best alignment, where k is a parameter defined by the user.
- ☐ Add support for fault tolerance: can your implementation tolerate a (single) node failure?
- ☐ Support multiple sequence alignment: rather than aligning a single sequence S into D, add support for multiple sequences ($S_1, S_2, \ldots$ in D) alignments.
- ☐ Your **own** idea: we support students pursuing their own interesting idea. In fact, we encourage teams to think of additional features they can add to a product showing the conceptual and engineering skills. If you think of an interesting feature, please discuss it with a TA first.

2.2 Deadlines

1. (mandatory) October 31, 2023 (week 44): Enroll in Canvas. Students who have not enrolled will not be graded and cannot receive credits for this course.
2. (mandatory) November 4, 2023 (week 44): Form a group and pick a lab project. Each project has a group created on Canvas, make sure all members enroll in that group.
3. (recommended) November 11, 2023 (week 45): Discuss and finalize your system's functional and non-functional requirements. It is recommended to discuss this with the supervisor.
4. (recommended) November 18, 2023 (week 46): Discuss with the supervisor your proposed design and plan for implementation.
5. (recommended) Before December 9, 2023 (week 49): Demonstrate to the supervisor the system implemented along with a 7-slide deck to enable yourself to be picked for the pitch session to be held December 13th.
6. (mandatory) Anytime before December 14, 2023 (week 50): Demonstrate to the supervisor the system implemented for the lab exercise.
7. (mandatory) December 23, 2023 (week 51): Turn in the lab-exercise report on canvas.
8. (recommended) December 13, 2023 (week 50): Demo Day for the selected groups.

To achieve the above-mentioned deadlines, your assignment can be divided into three parts:
Weeks 2—3:

- Analyze the system functionality and summarize its requirements.
- Analyze the (potential) system implementation and summarize your design's key features. Discuss with your team-mate(s) the alternatives for the design choices you have made, and summarize them in your report. Try to have the resulting requirements and design document (maximum 2 pages) approved by the supervisor before you start the actual implementation.

- Implement a part of the key features

Weeks 4—5:

- Finalize the design.
- Implement all key features.
- Test your system's implementation for different features: correctness, consistence, scalability etc.
- Per feature, consider how you are going to test its correctness and report on it.

Weeks 6—7:

- Implement and execute the test plans.
- Demonstrate your **working** prototype/product to the supervisor. Any feature that you claim it has, must be demonstrated. We recommend teams to make scripts that demonstrate each feature, to avoid having to setup everything during the demo. The demo can take at most 30 minutes, use this time wisely!
- Finish your report by including the achieved results and your conclusions, and send it to the supervisor.

Note: the estimated time required for the completion of this assignment (the large exercise) is 112 hours, equally divided by all team members.

Notes:

- Try not to exceed, for any experiment, 10 hours of work.
- You can leverage the same development and setup for several experiments. In this case, report the time spent for the shared part only for the first experiment and explain the large amount of time spent for it in the report.
- You can look at related scientific material to see how they tested their system, using something similar usually gives you a good basis.

3 Report Structure

The report should have the following structure:

1. *Report title, authors, and support cast* (lab supervisor and course instructors). For each person in your team, give name and contact information (email).
2. *Abstract*: a description of the problem, system description, analysis overview, and one main result. Size: one paragraph with at most 150 words.
3. *Introduction* (recommended size, including points 1 and 2: 1 page): describe the problem, the existing systems and/or tools about which you know (related work), the system you are about to implement, and the structure of the remainder of the article. Use one short paragraph for each.
4. *Background* on the application (recommended size: 0.5 pages): describe your product or the system your are working with (1 paragraph) and its requirements (1 paragraph per each of consistency, scalability, fault-tolerance, performance, etc.).

5. *System Design* (recommended size: 1.5 pages)
   - System overview: describe the design of your system or research, including the system operation, fault tolerance, and scalability components (which correspond to the homonym features required in this project).
   - (Optional, for bonus points, see Section 4) Additional (system) features: describe each additional feature, one sub-section per feature.
6. *Experimental Results* (recommended size: 1.5 pages)
   (a) *Experimental setup*: describe the working environments (DAS, Amazon EC2, etc.), the general workload and monitoring tools and libraries, other tools and libraries you have used to implement and deploy your system, and tools and libraries used to conduct your experiments.
   (b) *Experiments*: describe the experiments you have conducted to analyze each (system) feature, such as consistency, scalability, fault-tolerance, and performance. Analyze the results obtained for each system feature. Use one sub-section per experiment (or feature). In the analysis, also report:
      i. Service metrics of the experiment, such as runtime and response time of the service, etc. and describe what it measures.
      ii. (optional) Usage metrics and costs of the experiment where applicable.
7. *Discussion* (recommended size: 1 page): summarize the main findings of your work and discuss the tradeoffs inherent in the design of your system. Should you use a distributed system to implement the requested system? Try to extrapolate from the results reported in Section 6b for system workloads that are orders of magnitude higher than what you have tried in real-world experiments.
8. *Conclusion*
9. Appendix A: Time sheets (see Section 3.1)

3.1 Document the Time You Spend

You should report on the time it takes to conduct each major part of the assignment. Specifically, report:

- the `total-time` = total amount of time spent in completing the assignment (the large exercise).
- the `think-time` = total amount of time spent in thinking about how to solve the assignment (the large exercise).
- the `dev-time` = total amount of time spent in developing the code needed to solve the assignment (the large exercise).
- the `xp-time` = total amount of time spent in experiments for the assignment (the large exercise).
- the `analysis-time` = total amount of time spent in analyzing the results of the experiments for the assignment (the large exercise).
- the `write-time` = total amount of time spent in writing this report
- the `wasted-time` = total amount of time spent in activities related to the assignment (the large exercise), but which cannot be charged as think-time, dev-time, xp-time, analysis-time, or write-time.

4 Scoring

1. Quality of the report [9, 10] (presentation, style of writing, graphing, requirements analysis, design, analysis of results): +2,000 points.
2. Technical quality of the systems work (basic system features are supported): +2,000 points.
3. Bonuses
   - Making your source code open-source and your report public: +100 points.
   - Additional system features (feature and report): usually +125 points per feature (see Section 2.1).
   - Excellent report (writing, graphing, description of system): at most +250 points.
   - Excellent analysis (design of experiments, figures, analysis of results): at most +250 points.
4. Limits on bonuses
   - The total additional points for this exercise can amount to at most +1,000 points, representing additional features, excellent report and/or analysis bonuses, etc.
   - The additional features (see section 2.1) can amount to at most +500 points. Pay attention that some exercises limit the amount of bonus features that you can get credits for. Usually the amount of bonus points obtainable is equal per feature, but this may differ between assignments.

Note: writing a good technical report is scored equally to writing code for a system that works! This means that you should schedule time to write a good report, and for one revision to respond to our feedback.

5 Use of Existing Technology

You are not limited for this exercise to any technology, but:

- You CAN use the machines provided by our system DAS-4 or DAS-5, or by Amazon EC2 or another IaaS cloud (e.g. Azure, Google Cloud). For Amazon EC2, you can use the free resources provided through the Free Usage Tier [`http://aws.amazon.com/free/`]; if you use the m1.small (paid) instances of Amazon EC2, the estimated cost for this exercise does not exceed 8 EUR[1].
- You CAN use libraries that already provide a mandatory functionality required in Section C, as long as you implement a bonus feature listed in Section C in exchange AND after approval of the supervisor. If you only implement the mandatory functionalities, you CAN NOT use such libraries.

Examples of libraries that are fine to use when implementing only the mandatory functionalities are:

- Google RPC,
- Python's default communication libraries (sockets),
- Java Akka.

[1] At 8 Euro-cents per hour, assuming a total workload of 100 hours. Mileage may vary.

Examples of libraries NOT to use UNLESS you implement a bonus feature and you received approval of the supervisor are:

- Hazelcast, it provides fault-tolerance automatically,
- Frameworks such as Hadoop/Spark that automate fault-tolerance and load-balancing.

When using an external library, give due credit in your code and report (see also Section 6).
When in doubt about technology you intend to use, consult with the supervisor.

6 Anti-Fraud Policy (Zero-Tolerance)

Our anti-fraud policy is simple: zero-tolerance, within the limits set by VU Amsterdam. We will pursue each case of potential fraud, and will use to the maximum extent the means provided by VU Amsterdam to prevent, then to discover and punish (attempts to) fraud. You can learn more about how to prevent that you commit fraud via a discussion with the "studieadviseur", from Appendix 2 in the Regulations and Guidelines regarding examinations FEWEB [Amsterdam(2018)], or even from international sources such as Harvard's guidelines on avoiding plagiarism [Harvard(2019)].

References

[Amsterdam(2018)] VU Amsterdam. 2018. *Student Charter 2017-2018.* `https://www.vu.nl/en/Images/Studentcharter_2016_2017_tcm270-793620.pdf`

[Harvard(2019)] Harvard. 2019. *Avoiding Plagiarism.* `https://www.extension.harvard.edu/resources-policies/resources/tips-avoid-plagiarism`

[Smith and Waterman(1981)] T.F. Smith and M.S. Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 1 (1981), 195–197. `https://doi.org/10.1016/0022-2836(81)90087-5`