Graduation Project

Applied Data Science MSc

Optimising ASReview simulations: A generic multiprocessing solution for ‘light-data’ and ‘heavy-data’ users

Author: Sergei Romanov [1]

Student Number: 4725425

Supervisors: prof. dr. Rens van de Schoot [2] and Laura Hofstee [2]

Second Examiner: Ayoub Bagheri [2]

[1] Applied Data Science MSc. Student Department of Information and Computing Science, Faculty of Science, Utrecht University

[2] Department of Methodology and Statistics, Faculty of Social and Behavioral Sciences, Utrecht University

Abstract

Active learning (AL) aided systematic review pipelines are a promising tool for optimising and speeding up the performance of systematic reviews. In this paper, we present an architecture design with multiprocessing computational strategy for ASReview Makita workflow generator (Teijema, Van de Schoot et al., 2023) using Kubernetes software for the purposes of deployment with cloud technologies. The main goal of this study is to contribute to the following research in the study field of AL-assisted systematic review simulations by focusing on the parallel and distributed computing techniques. We provide an in detail technical explanation of the proposed cloud architecture and its usage manual. In addition to that, we conducted 1140 simulations studies investigating computational time required for ARFI Makita template using various number of CPUs and RAM settings. Our analysis demonstrates the degree to which ARFI Template can be accelerated with multiprocessing computing usage. The parallel computation strategy and the architecture design which were developed in the present paper can contribute to future research with more optimal simulation time and, at the same time, ensure the safe completion of the needed processes.

Table of contents:

1. Introduction

Scholars, scientists, journalists, and other types of researchers conduct systematic reviews, summarise various specific fields of knowledge – whether scientific or not – and extract overviews of relevant topics. While conducting a traditional systematic review implies marking pieces of text as relevant or irrelevant "by hand", the process of paper labeling can be dramatically optimised with the usage of different machine learning (ML) instruments – especially pipelines based on active learning (Van de Schoot et al., 2021).

In recent decades the research in the active learning (AL) field produced a lot of interest for various types of scientists, who study AL as a self-contained approach towards systematic reviews (Teijema, Seuren et al., 2023). In order to answer the questions posited by this new field of study, many computationally and data intensive simulations, software and statistical tests are required.

The computational intensity of these tasks may be influenced by the complexity of used machine learning (ML) tools. For example, some ML algorithms, such as neural networks (NN), by default could take 100-fold more time than simpler models (Teijema, Hofstee et al. 2023, p.8). Other ML models, for instance SVM, might have a square-increase in the training time, which depends on the number of input records (Ambert et al., 2013, p.12).

What is more, there might be another reason for this computational intensity. Running simulations multiple times to vary and test every component of the pipeline (e.g., feature extractor, sampling technique, ML model) becomes crucial to answer AL related research questions. For instance, it could imply an examination of the impact of different training datasets (Ferdinands et al. 2020), tests of ML models (Teijema, Hofstee et al. 2023), or evaluation of multiple datasets (Harmsen et al. 2021). Hence, the required number of needed computations exponentially increases as the components are varied, and it might reach more than 20 thousand simulations, as for example in the research of Teijema (2023).

One of contemporary AL packages, designed to ease such cases, provides a simulation template generator ASReview Makita (Make it Automatic), which can compose all the code that is necessary to execute hundreds or thousands of simulations (Teijema, Van de Schoot et al., 2023). Nonetheless, the current workflows generated by Makita pursue the sequential strategy of computations, so the simulation commands are executed one after another. If a lot of simulations are needed, this approach might result in unnecessarily vast timeframes due to queuing time, which can burden the research workflow.

However, such tasks require a large number of computations not because of intensive, but because of extensive computational complexity. Their ‘heaviness’ is caused not by algorithmically demanding tasks (i.e., in NN models), but rather by the high number of simpler ones. In other research fields, where a significant number of simulations is also employed for analysis, scholars adopt a parallel computing approach when the simulations are independent of one another – for example in physics (Neudorfer et al., 2012) or statistics (Lee & Kim, 2015).

In this study, we propose similar multiprocessing solution for AL aided systematic review simulations, which divides the large and complex set of simulations into several independent parts and optimises the computation time.

The current paper starts with a literature overview of parallel computing, containerisation and orchestration technologies, followed by a technical explanation of the proposed cloud architecture and a guideline for data scientists regarding the multiprocessing of Makita templates. Then, we present the results of a simulation study investigating computational time required for ARFI template using various numbers of CPUs and memory settings. The study ends with a discussion describing the limitations of the presented study and its potential development in the future research.

2. Research objectives

The main goal of this study is to contribute to the field of simulation studies investigating the performance of active learning (AL) aided systematic reviewing by focusing on the parallel and distributed computing techniques with usage of cloud environment and resources. By implementing such techniques, it is possible to considerably optimise computational time, making the research more efficient and less burdensome. For these purposes the research objective has been divided into two sub-questions: research goal (RG) and research question (RQ).

RG. How to optimise the simulations of AL pipeline with usage of modern parallel and high-performance computing techniques?

RQ. Can the usage of parallel computing on a cloud platform optimise the simulation time of an AL pipeline?

3. Literature review

3.1. Parallel and Distributed Computing

If some parts of the computational task (e.g. calculation, algorithm, or a combination of such) are performed simultaneously, such type of solutions is called parallel computing (or distributed computing, or multiprocessing) (Gottlieb & Almsi, 1989). The parallel computing is usually conflated with concurrent computing, but they are distinct although not mutually exclusive concepts. While the former advances the task performance by breaking it down into multiple similar and independent tasks, the latter approach also implements the "divide and conquer" principle. However, the tasks in concurrent computing are co-dependent and address different issues (Grossman & Anderson, 2012).

Nonetheless, both are utilised in modern computing methods. Their gain of popularity in computer architecture paradigms happens because they increase the performance of the processors, and thus reducing the heat production and power consumption (Asanovic et al. 2006).

3.2. Containerisation technology

Using a powerful local computer might be more recourse demanding than running simulations on cloud services for several times. For that reason, executing large processes on cloud might be a better option. To make the application scalable to the optimal number of instances, an approach called container-based virtualisation (containerisation) could be utilised. This technique involves packaging the application with all other technologies required for its functionality, making it compatible with any hardware environment (Scheepers, 2014).

The core principle of container technologies lies in enabling the processes and their resources to be isolated without any hardware emulation and specific hardware requirements (Scheepers, 2014, p.2). Another advantage of containerisation is that it makes software lightweight and portable by using its abstraction – a container image. It contains the information about necessary settings, code to install package dependencies, etc. (Kurtzer, Sochat, Bauer, 2017). Today there are multitude of open-source pieces of software developed for containerisation (i.e., Linux containers, Podman, Docker).

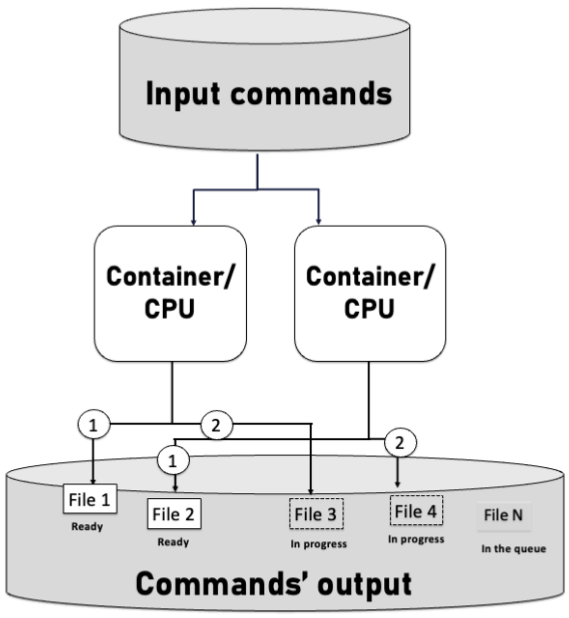
Containerised applications make it is possible to parallelise the simulations following the principle depicted on the Figure 1, which was inspired by the approach of Tesliuk et al.

(2019). Figure 1 shows an atomic example of two containerised instances parallelisation, that can be adjusted to a number of available CPU cores. Assigning one container with an instance of an AL software to one CPU unit enables the isolation of the inner processes from other CPUs. Further, we refer to a CPU or a CPU assigned to a container as CPU/Container.

In contrast to a sequential strategy of Makita generated workflow, the presented approach takes the queue of the necessary commands and creates a pool out of them. From that pool of "Input commands" we send them to various CPUs/Containers and execute them in parallel and simultaneously, thereby saving time. It is noteworthy that CPUs/Containers are not receiving a new command until they finish the previous one and put the output into an allocated file place, so there are as many queues as the number of CPUs/Containers.

Figure 1

The architecture of parallelising the AL-aided systematic review input simulation commands between two instances of CPU (or CPU attached to a Docker container) and allocating them to the files of commands' output



Note: After the Container/CPU finishes a simulation, it takes new command from the pool of input commands. Both left and right Containers/CPUs have their own queue, which is denoted by the numbers (e.g., 1 or 2). While the Files 1 & 2 are the output files of two already completed simulations, the Files 3 & 4 correspond to output of two simulations, which are in progress. File N is a set of files that are going to be created from upcoming commands.

For example, if we have the simulation study containing three simulation commands: simulation with parameters *x* and *y* (A), simulation with parameters *x* and *z* (B), simulation with parameters *y* and *z* (C), those are our input commands. When the execution is started, command A passes to the left CPU/Container (See Figure 1), and command B passes to the right CPU/Container. In the meantime, command C waits until the left CPU/Container provides the output of the command A to the File 1 and starts its execution only after the previous one is fully completed.

3.3. Orchestration System

While the traditional multiprocessing in algorithms utilise CPU kernel for allocating and scheduling tasks, the performance of the same functionality over containers requires special orchestration technologies. They were initially developed in large IT companies that have been implementing scalable highload services like Amazon, IBM, or Google. There are multiple available instruments of orchestration systems (i.e. Terraform, Ansible, Kubernetes).

Orchestration systems such as Kubernetes are a subset of system administration which can be distinguished by the possibility of automated configuring, coordinating, and managing of computer systems and software (Erl, 2005).

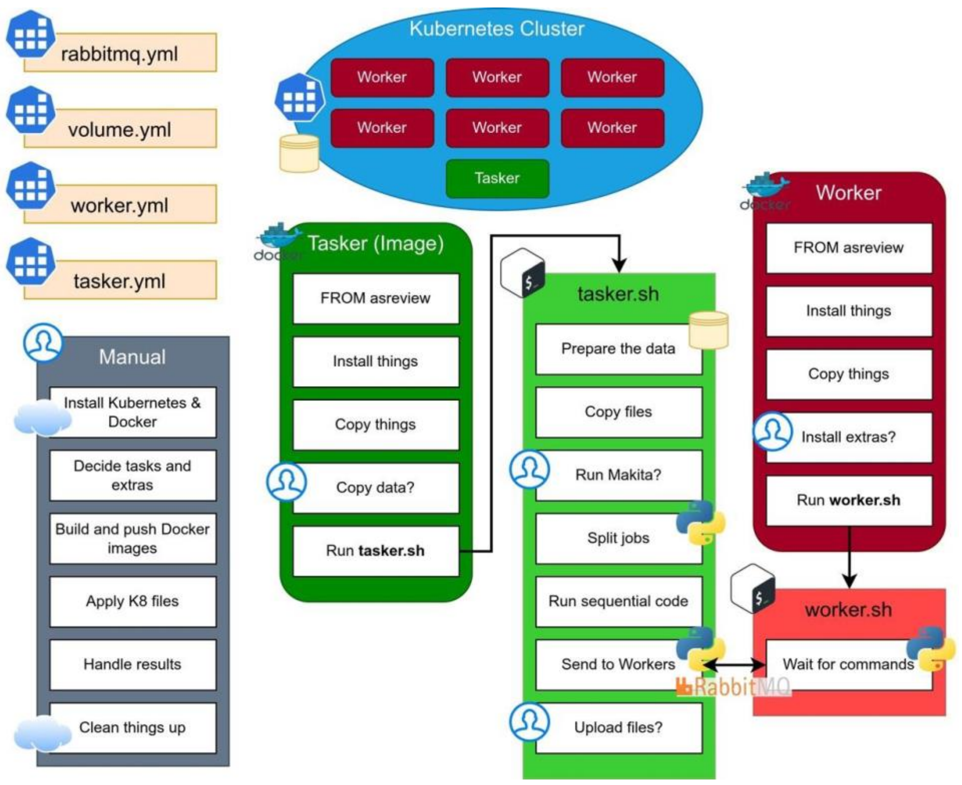
Orchestrating approach enables easy integration of diverse software in a unified computing platform and allows to manage and to scale containerised software in large computing infrastructures. In combination with container technologies, such management systems can ensure security of the parallel processes, their isolation from each other, and secure and simplified networking (Khan, 2017). This makes parallelisation on the level of containers safer than parallelisation on the level of CPUs themselves, for traditional computer kernel management is burdened by denial-of-service cases and paging out of processes (Kerrisk, 2012). Containers can guarantee an isolation of processes and troubleshooting functionality in the overload scenarios.

4. Orchestration System of ASReview and Types of Kubernetes Nodes

For **RG's** purposes this study presents cloud architecture based on ASReview implemented in Siqueira & Romanov (2023). This architectural design was devoted to applying parallel computation strategy to ASReview Makita templates (Teijema, Van de Schoot et al., 2023). This section is dedicated to explain and describe each of these components separately. The overall workflow diagram of the ASReview cloud architecture is presented on Figure 2. It presents a summary of the Kubernetes cluster and configuration of Kubernetes Nodes – a native abstractions in the Kubernetes architecture which can consist of either one of more application containers. Nodes may be two types (Jobs or Services), and the difference is in their persistence time.

Figure 2

Diagram of the ASReview parallelisation design for cloud Kubernetes cluster implementation that describes the setup manual steps and the way two Docker components (Worker and Tasker) communicate using a 'tasker.sh' and 'worker.sh' bash scripts with addition of RabbitMQ Message broker



Note: There are 4 types of components (RabbitMQ, Volume, Worker, Tasker), and there can be only one Tasker, Volume, RabbitMQ, whereas the Worker components can be multiplied as long as there is enough CPUs. Constituting a Kubernetes cluster, Tasker initiates the process using a given data and the tasker.sh, which distributes the Makita generated commands over the Workers. In response, after the simulation is completed, Worker stores the result in the Volume component and sends the message that is ready for a new simulation to Tasker component. The diagram was designed in Siqueira & Romanov (2023).

4.1.Message broker

To orchestrate all other components RabbitMQ software is used as a message broker. RabbitMQ implements the Advanced Message Queuing Protocol (AMQP) and supports various messaging patterns, such as request/reply, and what is more important for parallel processing – work queues (RabbitMQ). Message broker persists as a Service abstraction, meaning it is running until the user manually deletes it or it meets a deletion condition.

RabbitMQ solves the distribution of jobs (simulations commands in our case) among Worker nodes and prevents Workers from receiving a new job before the previous is solved by messaging about their completion. RabbitMQ utilises exchanges to route messages to working queues, and the Tasker node is subscribed to receive messages to 'know' when to send the next task.

4.2.Volume Node

The main and only functionality this component has to store the results and to send them to the user on demand. It persists as a Service abstraction.

4.3.Tasker Node

This component consists of several Makita scripts dockerised in one image and performs as a job Kubernetes abstraction, meaning it disappears after the given Makita commands are fully executed.

First, it copies the provided datasets and runs specified Makita template, which is split into three parts: the part of directories definition, the part of simulations commands, and the part of metrics computation. While the two latter are parallelised one after another to ensure that metrics can be computed based on the completed simulations, the former part cannot be parallelised as it creates the directories' structure, which must be executed sequentially.

After the directories are created, the Tasker node distributes the commands among Worker nodes via RabbitMQ – first, from the simulations part and, second, from the metrics part.

4.4. Worker Node

In comparison to the previous component, the Worker nodes are implemented as Kubernetes services. They are responsible for implementing the Makita templates' commands sent from the Tasker, and to store the results in the Volume node. In order to make it possible, each Worker Node, and thus container inside, has the list of packages and dependencies that ASReview application needs. Moreover, the Worker is responsible for sending messages back to Tasker, so it is aware when to send new commands.

4.5.Where to start

The presented architecture was provided with explicit documentation and the manual, which is also mentioned on Figure 2. The manual describes the steps necessary to run large number of simulations on SURF cloud environment. However, it can be implemented in other cloud and local environments.

This implementation and the manual have been made publicly available at the GitHub repository https://github.com/abelsiqueira/asreview-cloud, where all used scripts and documentation are covered under the Apache License Version 2.0 (Siqueira & Romanov, 2023).

5. Methodology

5.1. Set-up

Initially, the cloud infrastructure was developed and tested locally on several machines (Macbook M1 chip and Dell XPS 2018) using both Windows and MacOS operating systems for proof runs.

Then, we implemented our main time measurements using cloud environment with Linux Ubuntu OS. For cloud infrastructure provider we have been using SURF Cloud — noncommercial cloud infrastructure benefiting Dutch academic researchers and stakeholders (SURF). In Table 1, the specific details of the used software on cloud can be found.

Table 1

Software utilised during the study

Software	Name and version
Operating Systems	Windows (version 11), MacOS (version 13.2.1), Linux Ubuntu (version 20.04)
CPU and RAM	16 cores - 64 GB RAM
Cloud host	SURF Cloud[3]
Container software	Docker (version 24.0.0)[4]
Orchestration software	Kuberenretes, Minikube (version: 1.30.1)[5]
Active learning software	ASReview (version 1.2)[6]
Template generator for AL simulations	Makita (version 0.6.3)[7]
Message broker	RabbitMQ (version 3.12.0)[8]
Parallelisation Package	GNU parallel (version 20230522)[9]

Regarding the used AL tools, default ASReview simulation settings were utilised that consist of Naïve Bayes as a classifier model, TF-IDF as a feature extractor, 'MaxQuery' as a query strategy, and dynamic resampling as a balance strategy (ASReview LAB developers, 2023, Van de Schoot, 2020). With regard to Docker, Minikube Kubernetes and RabbitMQ, they were selected for the purposes of this study, since they are most-widely used, which makes the whole architecture more reproducible for following implementation.

5.2. Analytical strategy

The parallel computations might be implemented on various levels: bit-parallelism, instruction level, data level and tasks parallelism. Our multiprocessing strategy was focused on parallelising ASReview simulations on the task level, by using many container instances of the ASReview application and running separate tasks on them.

We conducted multiple time measurements of ARFI Makita template simulations using different limits for all allocated memory for Minikube Kubernetes implementation, different number of central power unit (CPU) cores, and different volumes of allocated RAM per CPU.

[3] SURF cloud, https://www.surf.nl

[4] Docker, https://www.docker.com

[5] Kubernetes Project, https://kubernetes.io/

[6] ASReview LAB developers. (2023). ASReview LAB - A tool for AI-assisted systematic reviews (v1.3a0). Zenodo. https://doi.org/10.5281/zenodo.7993446

[7] Teijema, J., Van de Schoot, R., Ferdinands, G., Lombaers, P., & De Bruin, J. (2023). ASReview Makita: a workflow generator for simulation studies using the command line interface of ASReview LAB (v0.6.3). Zenodo. https://doi.org/10.5281/zenodo.7838272

[8] RabbitMQ, https://www.rabbitmq.com

[9] Tange, O. (2011). Gnu parallel-the command-line power tool.; login: The USENIX Magazine, 36 (1): 42–47.

First, the default Kubernetes limit of RAM was 2GB for the whole implementation, then it was increased to 60 GB. Regarding CPU, the number of cores was varied between 1 and 14 with increment of 2 cores each test. Furthermore, we varied the memory allocated to each Worker container — one set of runs with 1024 megabytes of RAM per CPU and 2048 megabytes of RAM per CPU.

In addition to that, we used GNU parallel (Tange, 2011) package and implemented parallelised simulations on the same CPUs of SURF cloud machine, but without Docker and Kubernetes. In the current study we will refer to this implementation as 'bare metal'.

It is important to note that Kubernetes utilised additional CPUs by default for Tasker and for RabbitMQ one per each, and, thus, the run of Kubernetes cannot be compared to the 'bare metal' at the level of 16 CPUs; nonetheless, we included the measurements for 'bare metal' in order to show the maximum speed up capacity available with a given settings.

Thereby time measurements with CPUs number from 1 to 16 were implemented in four rounds and compared to a benchmark sequential ARFI run time of 387 seconds:

- Kubernetes with a default limit of RAM
- Kubernetes with 1024 megabytes of RAM per Worker
- Kubernetes with 2048 megabytes of RAM per Worker
- 'Bare metal' using GNU package.

5.3. Metrics

To evaluate the performance of multiprocessing strategies, time for computation, speedup ratio and parallel efficiency metrics were calculated per each run.

Time for computation: T denotes the time (in seconds) taken by a given simulation run from its start to finish.

Speedup ratio: $S_p = \frac{T_1}{T_p}$, where T_1 denotes the time for sequential computation of the whole set of simulations on one CPU (default run), and T_p is the time for computation of the whole set of simulations with parallelisation using *p* number of CPU cores (Eager, Zahorjan & Lazowska, 1989). Speedup ratio indicates how many times a given multiprocessed program run is faster than a sequential run. For example, if the serial process is run in T_1 = 10 seconds, and the same but parallelised process is run in 5 seconds, thus, the speedup ratio is 2.

Parallel efficiency: $E = \frac{S_p}{p}$, where S_p denotes the speedup for a certain number of cores, and p is the number of cores (Prasad et al. 2015, pp. 81- 82). This metric provides the information about the percentage of speedup each CPU core provides. If E < 1, speedup is called sublinear, if E ≈ 1, speedup is called linear, if E > 1, speedup is called superlinear.

5.4. Dataset

To conduct the time measurement, one dataset was taken from the Synergy datasets collection – a free and open-source dataset on study selection in systematic reviews, comprising 169,288 academic works from 26 systematic reviews (De Bruin et al. 2023). The dataset collection was published with an open CC0 1.0 Universal Licence. In our work we used PTSD dataset (Van De Schoot et al., 2017). It is a collection of 4,544 abstracts of studies related to post-traumatic stress disorder (PTSD) trajectories. It contains information on the title, authors, abstract, keywords, and record's status of relevance. This status indicates whether the study met the criteria for being included in systematic review. The overall inclusion rate (38 records, 8% of the whole data) and the ARFI template with 38 relevant records results in 1102 simulations overall with addition of 38 simulations of the benchmark (1140 in total). If we run ARFI per each combination of CPUs number with GNU (8×38) it is 304 simulations, and three times for Kubernetes with ARFI per each combination of CPUs number ($7 \times 38 \times 3$) is 798 simulations (38 + 304 + 798 = 1140) .

Moreover, the dataset of van de Schoot is used as a benchmark dataset in the ASReveiw software, meaning that it is more familiar to its current users, and it might be a feasible demonstration of the possibilities of cloud environment and architecture for them. This makes a good demonstration for the purpose of the current study in the available time frame.

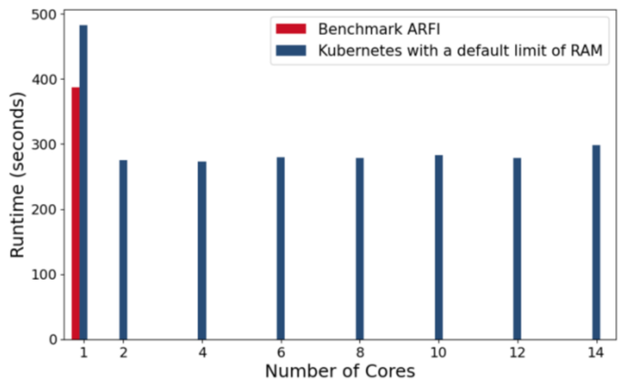
6. Results

6.1. Performance Analysis and Limitations of the Default Kubernetes Settings

During the initial time measurements of 266 simulations with Kubernetes, it was found that after using more than 4 CPUs and reaching the performance of 273 seconds the computational time (*T*) ceases to decrease. With accordance to Figure 3, *T* stays between 275 and 298 seconds. That was caused by default setting of Minikube Kubernetes, which limits the RAM memory for all Worker containers to 2 GB, and the usage of more than 4 Worker nodes with 1024 megabytes of RAM conflicted with these limitations. Although the default settings have imposed some limits, the usage of such amount of memory already gave almost a two-fold speedup from 483 seconds (non-parallelised default ARFI depicted by the red bar) to a minimum 273 seconds in the first round of tests. Because those Minikube limits do not provide the information about the accelerating capacity of the architecture, speedup ratio and parallel efficiency were not computed for this round.

Figure 3

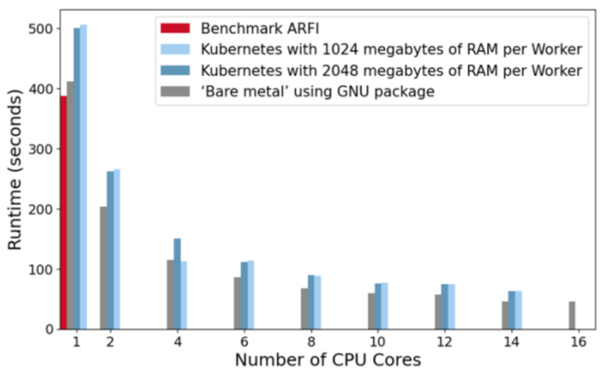
Multiprocessing timings (in seconds) for 38 parallel simulations distributed over various CPUs numbers (from 1 to 14 CPUs) for one round of runs and a benchmark timing.



Note: Each bar is time of running a set of 38 simulations. Kubernetes needs CPU per Tasker and RabbitMQ components, hence there is no Kubernetes 16 CPUs results (See Appendix A). There are 304 simulations in total on this plot.

Figure 4

Multiprocessing timings (in seconds) for 38 parallel simulation studies distributed over various CPUs numbers (from 1 to 16 CPUs) for 3 rounds of runs and a benchmark timing



Note: Each bar is time of running a set of 38 simulations. Kubernetes needs CPU per Tasker and RabbitMQ components, hence there is no Kubernetes 16 CPUs results (See Appendix A). There are 874 simulations in total on this plot.

6.2. Comparison of Parallel Computing Strategies

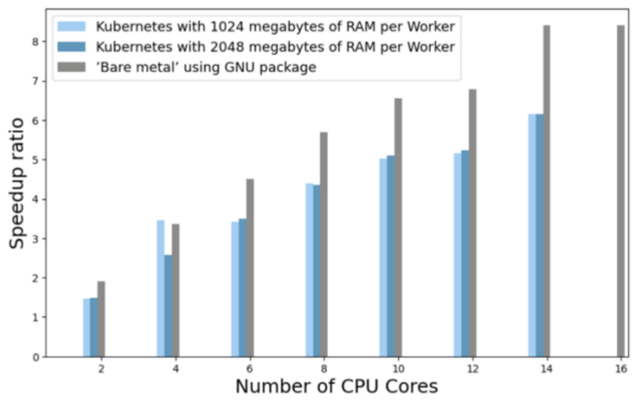
After adjusting the settings limits and allocating more RAM to each CPU, the results indicated a decrease in overall computational time as well as per each CPU number (see Figure 4). Whereas with only one CPU the default ARFI run (387 seconds) was faster than 'bare metal' (411 seconds) and both second (506 seconds) and third (500 seconds) Kubernetes runs, with the step to 2 CPUs the latter three had *T* of 203, 265 and 262 seconds, respectively, and outperformed all first Kubernetes runs from Figure 3 (273-483 seconds) and the default ARFI run.

Comparing Kubernetes runs with different RAM allocation per Worker, the third round of simulations with 2048 megabytes of allocated RAM per Worker had a different outcome than with 1024 megabytes: it was faster (the mean difference is approximately 3.11 seconds), but still slower than the "bare metal" simulations.

What is more, all implementations apart from the first Kubernetes round show a more drastic speedup in the steps between smaller number of CPUs than with 12, 14 or 16 CPUs, thereby forming a plateau in terms of time optimisation with the higher numbers of CPUs (see Figure 5). For example, for GNU implementation there are similar timings of 46 seconds when 14 and 16 CPUs are used, resulting in the same speedup. At the same time the speedup between 12 and 14 CPUs for both Kubernetes was 5.16 and 5.22 times and 6.14 and 6.14 times, respectively, which is faster than benchmark ARFI run.

Figure 5

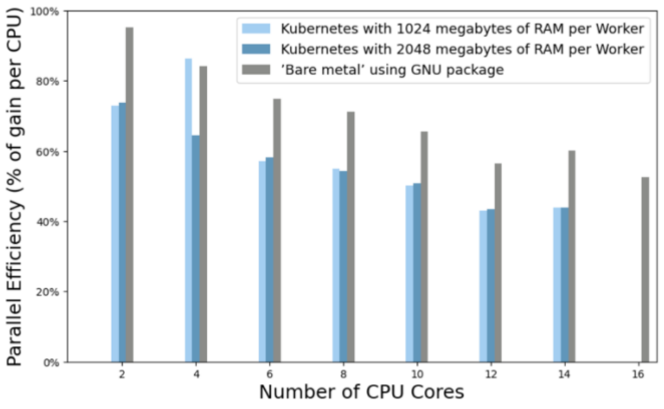
Speedup ratios for 38 parallel simulation studies distributed over various CPUs numbers (from 1 to 16 CPUs) for 3 rounds of runs



Note: On the x-axis, there are numbers of CPU cores, and on the y-axis, there are ratios of the benchmark run to the 3 rounds of runs, respectively. Also, Kubernetes needs CPU per Tasker and RabbitMQ components, hence there is no Kubernetes 16 CPUs results (See Appendix B), and the simulations with 1 CPU were not included because they do not have parallelising aspect. Each bar is time of running a set of 38 simulations. There are 570 simulations on this plot in total.

Figure 6

Parallel Efficiency for 38 parallel simulation studies distributed over various CPUs numbers (from 1 to 16 CPUs) for 3 rounds of runs



Note: On the x-axis, there is number of CPU cores, and on the y-axis, there is a percentage of speedup (efficiency) each core has in the 3 rounds of runs. Also, Kubernetes needs CPU per Tasker and RabbitMQ components, hence there is no Kubernetes 16 CPUs results (See Appendix C), and the simulations with 1 CPU were not included because they do not have parallelising aspect. Each bar is time of running a set of 38 simulations. There are 570 simulations on this plot in total.

In contrast, the initial step from 2 to 4 CPUs produced a larger speedup: for GNU – from 1.9 to 3.36 times, for Kubernetes with 1024 megabytes – from 1.46 to 3.45 times, for Kubernetes with 2048 megabytes – from 1.47 to 2.58 times (See Figure 5).

6.3. Speedup Patterns and Efficiency

It is noteworthy that already in the 2 CPU run the GNU parallelisation solution achieved 95.3% parallel efficiency, allocating each core with maximum observed efficiency (see

Figure 6 and Appendix C). In comparison, Kubernetes showed only 73.8% efficiency in similar CPU configurations. Throughout the whole study Kubernetes indicated a lower level of parallel optimisation than GNU, except for the case of one run with 4 CPUs and 1024 of RAM, when it was faster and more efficient by 2%.

With more than 12 cores, in both cases Kubernetes exhibited less than 50% efficiency (see Figure 6 and Appendix C), indicating diminishing returns in terms of performance improvement, while GNU had 52% of parallel efficiency per core only in the last 16 CPUs run.

The overall efficiency and speed of all rounds of simulations showed a notable decline, accelerating fewer times than the number of CPUs, constituting a sublinear manner of speedup. For instance, the most optimal and efficient simulation run improved by 1.903 times with 2CPUs in contrast to the serial ARFI run, which was the closest run to a linear speedup (2 times faster with 2 CPUs). Nevertheless, it was discovered that all parallel computing strategies implemented in this study perform faster than a default sequential ARFI run (387 seconds) (see Figures 3 & 4).

7. Discussion

We presented an architecture design with multiprocessing computational strategy for ASReview Makita (Make It Automatic) templates (Teijema, Van de Schoot et al., 2023) which help running the simulations studies mimicking the screening process of an active learning (AL) aided systematic review. The provided solution can be conducted on both local and virtual machines. The number of Kubernetes Workers is only limited to the availability of CPU and has less queuing computational limits than classical sequential strategies, which suits the necessities of processing numerous systematic review simulations mentioned in the research goal (**RG)**.

Moreover, we demonstrated the degree to which Makita ARFI template can be accelerated with multiprocessing usage. In comparison to serial processing benchmark, all four rounds of paralleled runs of ARFI were executed faster just using more than one CPU. With suggested parallel computation technique developed with Kubernetes architecture design (Kubernetes), future research can benefit with more optimal simulation time and, at the same time, ensure the safe completion of the needed processes.

7.1. Comparing Kubernetes' and 'Bare metal' GNU's accelerations

In previous studies, which have also implemented orchestration solutions for computationally intensive tasks (Tesliuk et al., 2019), Kubernetes was compared to a parallelisation without usage of containers and orchestration ('bare metal'). This "bare metal" implementation addresses the evaluation of parallelisation raised in research question (**RQ)**.

However, initial expectations based on Tesliuk et al. (2019), where Kubernetes demonstrated faster and more efficient performance, were not met during this study. The overperformance of Kubernetes over GNU was observed only with 4 cores with 1024 megabytes of RAM per Worker, suggesting that this specific configuration may be optimal for Kubernetes with the given settings and dataset. Nevertheless, it still shows an opposite outcome compared to results of Tesliuk et al. (2019, p. 70), and it is noteworthy that this optimal configuration may not be generalisable to other datasets or scenarios.

There are several differences in the set-ups of our analysis and their work, which potentially might have produced such outcome: the study used GPU, not CPU, the data was not textual but particles' images, and there was no Message broker software (Tesliuk et al., 2019, pp. 67-68). In this study we suppose that the presence of RabbitMQ within the ASReview Kubernetes setup has affected its overall performance, preventing it from outperforming the "bare metal" solution, as difference in GPU and data types would have affected processing time regardless of containerisation or 'bare' CPU exploitation.

Although the parallelisation with GNU package outperformed Minikube Kubernetes, except for one specific test, both address **RQ** with possibility of making situations of AL usage faster and more optimal. Despite "bare metal" solution provided higher speedup and parallel efficiency, it should be noted that it lacks the degree of security and process isolation possessed by containers (Khan, 2017, p. 44). In more data-intensive cases, for which the cloud architecture was designed, parallelising on default CPUs may compromise process security and introduce troubleshooting challenges, such as process page-out (Kerrisk, 2012). Thus, both the Kubernetes architecture or GNU parallel package may be used for optimisation of ASReview Makita, thereby constituting a trade-off between the degree of optimisation and the security of data-processing.

7.2. Limitations

The dataset we analysed possesses its own specific and idiosyncratic textual features, and this influences the computational time and number of cores with the best efficiency. In other words, the provided results may lack generalisation; however, the time measurements and experimental runs demonstrate the potential behind parallelising simulation runs in Makita template.

The conducted experiments methodologically are not suitable for drawing causal conclusions about the efficiency of the presented strategy, but rather present observational and descriptive results. Cloud and multiprocessing technologies should not be perceived as a 'silver bullet' solution for acceleration of ASReview simulations, because the performance of pipeline simulations varies dramatically depending on the choice of dataset (Teijema, 2023).

7.3. Further research

High variability of AL performance with dependance on data brings new challenges and questions about parallel simulations of systematic reviews. The datasets' characteristics are different not only in size and proportion of relevant records, but also in the semantics of the covered topics, syntax, vocabulary, and morphology of the datasets' language (Kroft, 2022). Continuing the current research, it is noteworthy to focus on the extensions in terms of data in the similar and other settings of Makita templates.

Moreover, many of the feature extractors and classifiers, that are utilised in AL pipelines, support parallelisation by design. Whereas random forest can be parallelised (Chen et al., 2016), but does not take a lot of time, most of neural networks (NN) architectures tend to have the longest learning time frames (Teijema, Hofstee et al, 2023, p.8). Expanding on the parallelisation aspect, it should be noted that NN are specifically suitable to be parallelised on (graphical power units) GPUs (Keckler et al., 2011). The parallelisation on GPUs can bring about significant benefits, particularly in feature extraction or classification timeframes with NN architectures.

Furthermore, exploring GPU parallelisation could offer valuable insights into parallelising at lower bit-levels. Since the simulation conducted at the task level resulted in a sublinear speedup, it would be interesting to investigate whether a different level of parallelisation can provide greater efficiency.

7.4. Impact on the field

Moving on to the impact on the field, the results obtained from this analysis can contribute to the research process of following studies in the field of assisted systematic review pipelines: first, as a software for optimal simulations, second, as a basis for following studies of multiprocessing effects on simulations' speedup.

While the direct impact on non-datascience users of AL might be limited, except for the progress in the software development, there might be indirect effects that ought to be considered. Employing more efficient parallel computing techniques leads to a relative reduction in energy consumption, which aligns with the emergence of the 'green' cloud computing paradigm in recent decades (Kumar & Buyya, 2012). Advocating for a more efficient and potentially environment-friendly solution not only benefits users but also has a broader effect on the public at large.

Overall, the usage of parallelisation techniques in the simulation of AL pipelines has the potential to greatly enhance computational efficiency. While parallelising simulations without using additional technologies turns out to be the fastest implementation, the pursue of the containerised software strategy ensures safely isolated processes and computing, and, at the same time, saves considerable amount of time. By balancing between the safe computation and optimisation further research in the AL-aided systematic review can advance with the software design presented in this study.

8. Open Science Statements: Data, Code Availability and Generative AI usage

This study did not deal with personal information in any way. For this study, we used data, which was published under of Apache 2.0 license (Van De Schoot et al. 2017).

This study has been made publicly available at https://github.com/zoneout215/asreview-makita-multiprocessing, where all used scripts and generated data have been made freely available under the MIT licence. Please note that the GitHub repository is maintained and updated by the author, and it may contain more recent versions or enhancements of the code used in this study.

Throughout the work on this study, generative AI tools, including OpenAI's ChatGPT and Copilot development environment extension, were used. These were supplementary aids, and all final interpretations and content decisions were the sole responsibility of the author.

Acknowledgements

I would like to express my deepest gratitude to my parents, Sergei and Evgeniia Romanov, whose unwavering support, love, and belief in me have been the driving force behind my accomplishments. To my girlfriend, Jenet Abdullaeva, your constant encouragement and understanding have been my rock throughout this journey. I am immensely grateful to my supervisors, Prof. Dr. Rens van de Schoot and Laura Hofstee MSc, for their invaluable guidance, expertise, and mentorship. Special thanks to Abel Soares Siquera for his contributions and sharing of expertise and whole ASReview Team for their exceptional support. Lastly, I extend my appreciation to my fellow Applied Data Science students for their camaraderie and shared experiences. To all mentioned and those who have supported me, thank you for being instrumental in the completion of my thesis and my personal growth.

Literature

Ambert, K. H., Cohen, A. M., Burns, G. A., Boudreau, E., & Sonmez, K. (2013). Virk: an active learning-based system for bootstrapping knowledge base development in the neurosciences. Frontiers in neuroinformatics, 7, 38.

Ansible, https://www.ansible.com/

Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., ... & Yelick, K. A. (2006). The landscape of parallel computing research: A view from berkeley.

ASReview LAB developers. (2023). ASReview LAB - A tool for AI-assisted systematic reviews (v1.3a0). Zenodo. https://doi.org/10.5281/zenodo.7993446

Chen, J., Li, K., Tang, Z., Bilal, K., Yu, S., Weng, C., & Li, K. (2016). A parallel random forest algorithm for big data in a spark cloud computing environment. IEEE Transactions on Parallel and Distributed Systems, 28(4), 919-933.

De Bruin, J., Ma, Y., Ferdinands, G., Teijema, J., & Van de Schoot, R. (2023). SYNERGY - Open machine learning dataset on study selection in systematic reviews. DataverseNL. https://doi.org/10.34894/HE6NAQ

Docker, https://www.docker.com

Eager, D. L., Zahorjan, J., & Lazowska, E. D. (1989). Speedup versus efficiency in parallel systems. IEEE transactions on computers, 38(3), 408-423.,

Erl, T. (2005). Service-Oriented Architecture: Concepts, Technology & Design. Prentice Hall.

Ferdinands, G., Schram, R., Bruin, J. D., Bagheri, A., Oberski, D. L., Tummers, L., & Schoot, R. V. D. (2020). Active learning for screening prioritization in systematic reviews-A simulation study.

Gottlieb, A., & Almsi, G. (1989). Highly parallel computing.

Grossman, D., & Anderson, R. E. (2012, February). Introducing parallelism and concurrency in the data structures course. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (pp. 505-510).

Harmsen, W., de Groot, J., Harkema, A., van Dusseldorp, I., De Bruin, J., Van den Brand, S., & Van de Schoot, R. (2021). Artificial intelligence supports literature screening in medical guideline development: towards up-to-date medical guidelines.

Keckler, S. W., Dally, W. J., Khailany, B., Garland, M., & Glasco, D. (2011). GPUs and the future of parallel computing. IEEE micro, 31(5), 7-17.

Kerrisk, M. (2012). LCE: The failure of operating systems and how we can fix it. LWN.net. https://lwn.net/Articles/524952/

Khan, A. (2017). Key characteristics of a container orchestration platform to enable a modern application. IEEE cloud Computing, 4(5), 42-48.

Kroft, M. V. D. (2022). Language morphology in active learning aided systematic reviews (Master's thesis).

Kubernetes, https://kubernetes.io/

Kumar, S., & Buyya, R. (2012). Green cloud computing and environmental sustainability. Harnessing green IT: principles and practices, 315-339.

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. PloS one, 12(5), e0177459.

Lee, J., & Kim, S. (2015). On speeding up stochastic simulations by parallelization of random number generations. Computers & Mathematics with Applications, 70(5), 1018-1026. https://doi.org/10.1016/j.camwa.2015.06.014

Linux containers, https://linuxcontainers.org

Neudorfer, J., Stock, A., Schneider, R., Roller, S., & Munz, C. D. (2012). Efficient parallelization of a three-dimensional high-order particle-in-cell method for the simulation of a 170 GHz gyrotron resonator. IEEE Transactions on Plasma Science, 41(1), 87-98.

Podman, https://podman.io/

Prasad, S., Gupta, A., Rosenberg, A., Sussman, A., & Weems, C. (2015). Topics in Parallel and Distributed Computing. Morgan Kaufmann.

RabbitMQ, https://www.rabbitmq.com

Scheepers, M. J. (2014, June). Virtualization and containerization of application infrastructure: A comparison. In 21st twente student conference on IT (Vol. 21).

Siqueira, A., Romanov S. (2023). Asreview-cloud, repository Retrieved from https://github.com/abelsiqueira/asreview-cloud

SURF cloud, https://www.surf.nl

Tange, O. (2011). *Gnu parallel-the command-line power tool.;* login: The USENIX Magazine, 36 (1): 42–47.

Teijema, J. J. (2023). jteijema/asreview-simulation-project: v1.1.4 (v1.1.4). Zenodo. https://doi.org/10.5281/zenodo.7993561

Teijema, J. J., Hofstee, L., Brouwer, M., De Bruin, J., Ferdinands, G., De Boer, J., ... & Bagheri, A. (2023). Active learning-based Systematic reviewing using switching classification models: the case of the onset, maintenance, and relapse of depressive disorders. Frontiers in Research Metrics and Analytics, 8.

Teijema, J. J., Seuren, S., Anadria, D., Bagheri, A., & van de Schoot, R. (2023, June 29). Simulation-based Active Learning for Systematic Reviews: A Systematic Review of the Literature. Retrieved from psyarxiv.com/67zmt

Teijema, J., Van de Schoot, R., Ferdinands, G., Lombaers, P., & De Bruin, J. (2023). ASReview Makita: a workflow generator for simulation studies using the command line interface of ASReview LAB (v0.6.3) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.7838272

Terraform, https://developer.hashicorp.com/terraform

Tesliuk, A., Bobkov, S., Ilyin, V., Novikov, A., Poyda, A., & Velikhov, V. (2019, December). Kubernetes container orchestration as a framework for flexible and effective scientific data analysis. In 2019 Ivannikov Ispras Open Conference (ISPRAS) (pp. 67-71). IEEE.

Van De Schoot, R., De Bruin, J., Schram, R., Zahedi, P., De Boer, J., Weijdema, F., ... & Oberski, D. L. (2021). An open source machine learning framework for efficient and transparent systematic reviews. Nature machine intelligence, 3(2), 125-133.

Van De Schoot, R., Sijbrandij, M., Winter, S. D., Depaoli, S., & Vermunt, J. K. (2017). The GRoLTS-checklist: guidelines for reporting on latent trajectory studies. Structural Equation Modeling: A Multidisciplinary Journal, 24(3), 451-467

Appendix

Appendix A
Time for computation measurements

Number of cores	K8s_test1	K8s_test2	K8s_test3	bare_metal
1	483	506	500	411
2	275	265	262	203
4	273	112	150	115
6	280	113	111	86
8	279	88	89	68
10	283	77	76	59
12	278	75	74	57
14	298	63	63	46
16	0	0	0	46

Appendix B
Speedup ratio measurements

Number of cores	speedup_test1	speedup_test2	speedup_test3	speedup_metal
2	1.460377	1.460377	1.477099	1.906404
4	3.455357	3.455357	2.580000	3.365217
6	3.424779	3.424779	3.486486	4.500000
8	4.397727	4.397727	4.348315	5.691176
10	5.025974	5.025974	5.092105	6.559322
12	5.160000	5.160000	5.229730	6.789474
14	6.142857	6.142857	6.142857	8.413043
16	0	0	0	8.413043

Appendix C
Parallel efficiency measurements

Number of cores	efficiecy_test1	efficiecy_test2	efficiecy_test3	efficiecy_metal
2	0.730189	0.730189	0.738550	0.953202
4	0.863839	0.863839	0.645000	0.841304
6	0.570796	0.570796	0.581081	0.750000
8	0.549716	0.549716	0.543539	0.711397
10	0.502597	0.502597	0.509211	0.655932
12	0.430000	0.430000	0.435811	0.565789
14	0.438776	0.438776	0.438776	0.600932
16	0	0	0	0.525815