Notes on CS BSc. and MSc. Thesis Content at VU Amsterdam

Animesh Trivedi
a.trivedi@vu.nl
https://animeshtrivedi.github.io/
version: v1.5

May 19th, 2021

Contents

Version History

- v1.5 : Added references and tex template link to the Reproducibility Appendix.
- pre-v1.4 : Initial versions.

What is this about

Over the past years I have given a common subset of advice regarding how to write your thesis to multiple BSc and MSc students. This document is a synthesis of these advises, dos and don'ts that I have told them. In the coming section, I will add details of the kind of content you should be covering in your thesis and what I expect to see in each section of your thesis. BSc and MSc theses differ in the scope of the problem that they cover. Also, these are a broader set of advice that will also help when you are writing a research paper.

I am happy to have input from you in this matter and let me know if I have missed something that you covered in your thesis, and/or how we can improve the instructions here. This is not meant to be a comprehensive set of instructions, and following these guidelines does not imply getting the maximum grade.

General Guidelines

- Aim to produce a first complete draft 2-3 weeks before the final hand-in deadline. The earlier you give me a draft to review, more detailed feedback you will get.
- Spell check and proofread your draft before you give me. Do not leave half broken sentences and empty subsections.
- Aim to write frequently, and ask questions early.
- You will have one major and one minor revision opportunity before you hand in your final thesis draft for grading. It is unreasonable for me to read and re-read your drafts every week, hence, aim to write a first as complete draft as possible for maximum feedback.
- When writing **"Always think like a compiler - have I explained and/or defined all the ideas that I am about to use?"**. It is a responsibility of the writer to write an easy to read document, not the other way around. Do not just write for the writing sake to fill up pages. Try to make a simple, linear structure of your thesis. Build your case step by step, one new idea at a time, one new configuration or complexity. Try to reduce the "cognitive load" of your thesis as much as possible. If you want to test what does *"cognitive load"* mean, then try to read your own thesis very late in the night when you are really tired or fresh in the morning when you do not remember all the details up front. Re-read what you have written after a couple of days to see if you remember what is the key message that you were trying to get across.
- Always read and re-read your own thesis before you submit for a review.
- Your thesis PDF is for everyone else than you and me. Think of your friend who also studies Computer Science but does not know what you work on. They are not invested in your problem, and are not talking to you over the last couple of months. Hence, think and write down all the details which are necessary for them to understand what you have done.

English Usage

Pay attention to your English usage, especially the use of a paragraph and subsection. Each paragraph is supposed to cover one logical idea. Do not ramble. Do not cover 100 different possibilities and ideas in a single paragraph. After each paragraph, consider does it explain one idea and one idea only cleanly. Do not write paragraphs which are 1 or 2 pages. For every paragraph connect it back and forward. Similar advice goes for subsections. Think, if the heading of the subsection makes sense. If I hide the title of your subsection, would a reader come up with a similar heading if she or he reads the content of your subsection. For every new idea: write what you are about to explain, explain it, and recap what you have explained.

Do not use colloquial terms in your writing. Read papers and see how they present information and ideas. Do not use terms like "may", "could", "perhaps" - try to write as cleanly as possible without ambiguity. Always write in the present tense in active voice. Do not use frivolous adjectives and/or adverbs like "very much", "tremendous", "very fast", "high overheads". Always aim to quantify. What does "fast" or "slow" mean, can you measure it? in what term, what context? Be as specific as possible when you report information.

The golden rule of writing is (follow this for each subsection, section, and chapter) (i) Write what you are going to tell the reader and why is it important; (ii) Write and explain it to the reader; (iii) Recap what you have just explained to the reader.

Mandatory Reading List

Please read the following references carefully as they cover many of ideas which are iterated throughout this document:

1. George D. Gopen & Judith A. Swan, The Science of Scientific Writing, https://github.com/animeshtrivedi/notes/blob/master/docs/the-science-of-scientific-writing.pdf.

2. Gernot Heiser, Tips and Guidance for Students Writing Papers and Reports, https://www.cse.unsw.edu.au/~gernot/style-guide.html.

3. Gernot Heiser, Systems Benchmarking Crimes, https://www.cse.unsw.edu.au/~gernot/benchmarking-crimes.html.

4. John Ousterhout, Always Measure One Level Deeper, https://m-cacm.acm.org/magazines/2018/7/229031-always-measure-one-level-deeper/fulltext.

5. Roy Levin, and David D. Redell, How (and how not) to write a good systems paper, (applicable to your thesis work as well), https://www.usenix.org/conferences/author-resources/how-and-how-not-write-good-systems-paper.

6. Inclusion and Diversity in Writing, https://acmsocc.github.io/2020/inclusion_and_diversity_in_writing.html.

In the End

Take time to develop writing skills. It is going to be good from one day to another. It is as important as learning how to code, if not more. Write drafts and seek feedback. You can be creative in your writing once you have mastered the basics.

Most importantly, your thesis work is probably the biggest piece of work you have done so far in your studies. Care for it. Take pride in your work and writing - your name is attached to it!

1 Abstract

An abstract is a compressed or zip summary of your thesis. You can follow a simple structure:
(i) the broader societal, economic, scientific context, who are the people who might be interested in your work (2-4 sentences);
(ii) what is changing and what broad problem(s) does this change creates, has the problem always been here or something has triggered this problem, is it a specific problem or part of a bigger trend in your field (2-4 sentences);
(iii) what specific scientific/research problems that this thesis focuses on (4-6 sentences);
(iv) how does this thesis work tackle the problem (4-6 sentences);
(v) Key findings, give "quantitative results". For example, `Our results show that our system decreases SQL query execution time by 50.6% in the cloud` (1-2 sentence);
(vi) How can I use this thesis work. For example, `The code, and data is available at https://...`

These parts are guidelines, not strict rules (except the last one). Based on the flavor of the thesis you do, you will budget your word/sentence quotas differently.

2 Introduction

Introduction is an unzipped version of Abstract. Keep in mind that the Introduction and Conclusion *should* (aim for it) be broadly readable by all of your scientific peers, e.g., friends from Physics or Mathematics. Use simple, general language and gently introduce your thesis to the world. Where does this thesis work fit in, who are the people for whom this could be useful work, how much societal or economic impact does this research field have, what are the bigger trends in the area, etc. Introduction should be very broadly readable. For example, you could be doing something in the domain of storage where you talk about our societies being digital, and producing vast quantities of data (200 Zettabytes by 2025), how this data is generated, stored and processed, who are the people and companies who are interested in building fast storage systems, etc.

2.1 Context

Context is more specific and more specific to your research direction. Here you can gradually use more computer science language and terminology. What specific computer science topic are you working on? What is happening in this area? What are the new advancements? How are these advancements (or lack of thereof) What specifically is the problem, why is it a problem now, is it something new or an old problem in a new context, are you building something to try out something new, etc.

At the end of your context it should be very clear what are the broad challenges in this domain of your research. You should even summarize it clearly at the end of this section.

2.2 Problem Statement

What is the very specific technical and scientific problem that you are addressing in this work? Why is this problem important? Is this problem timely? What is the impact of the problem? What happens if it is not solved? What kind of a new class of systems should solving this problem will enable? Problem statement section is very specific to your thesis problem.

2.3 Research Questions

Research questions are scientific questions that you are doing to answer in this thesis work. There are many flavors of scientific questions, however, in a typical "systems" thesis work you will be designing and prototyping a system and benchmarking it. Hence, your scientific questions would be exploring questions in the scope of (i) what is the right design and/or how to come up with a right design?; (ii) what are the key challenges, or how to identify right parameters for a certain processes; and/or (iii) how fast, slow, efficient, etc. can a new system be?

A typical way to think about scientific questions is that you should be able to answer questions either qualitatively (by gathering literature evidence, complexity

analysis, taxonomies, design processes, designing and implementing concepts) or quantitatively (by running experiments and measuring). See Conclusion section about how to answer the research questions in more detail.

Always justify why your research question is important, and how do you plan to answer them, and what their answers will enable us to do that was not possible yet.

2.4 Research Methodology

How do you plan to answer the research questions? There are typically few accepted research methodologies that you should be following and referring in your work, which are:

- **(Methodology M1)** Quantitative research (statistical modeling, simulations, comprehensive surveys) [1-2];
- **(M2)** Design, abstraction, prototyping [3, 4, 5];
- **(M3)** Experimental research, designing appropriate micro and workload-level benchmarks, quantifying a running system prototype [6, 7, 8];
- **(M4)** Use-case study, collecting operational traces, collaborating with users and partners (following best practices [9, 10, 11]);
- **(M5)** Open-science, open-source software, community building, peer-reviewed scientific publications, reproducible experiments [12, 13, 14, 15].

Only include the ones you have used in your thesis work and reference them where you have used them. So for example, if you have done M3 and M5 and then include them and their references. Then in your text M3 will become M1 and M5 will become M2. Rename them appropriately.

References:

1. Naim A. Kheir, Systems Modeling and Computer Simulation,(2nd ed.). Marcel Dekker, Inc., NY,USA.
2. Yair Levy, Timothy J. Ellis, (2006) A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research. Informing Sci J 9: 181-212.
3. Alexandru Iosup, Laurens Versluis, Animesh Trivedi, Erwin Van Eyk, Lucian Toader, Vincent vanBeek, Giulia Frascaria, Ahmed Musaafir, Sacheendra Talluri (2019) The AtLarge Vision on the Design of Distributed Systems and Ecosystems. ICDCS 2019: 1765-1776.
4. R. Hamming, The Art of Doing Science and Engineering: Learning to Learn, CRC Press, 1997.
5. K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, A Design Science Research Methodology for Information Systems Research, Journal of Management Information Systems 24(3): 45-77 (2008).

6. R. Jain (1991) The Art of Computer Systems Performance Analysis. John Wiley & Sons Inc., NewYork, USA.

7. Gernot Heiser (2019) Systems Benchmarking Crimes. http://www.cse.unsw.edu.au/~Gernot/benchmarking-crimes.html, 2020.

8. John Ousterhout. 2018. Always measure one level deeper. Commun. ACM 61, 7 (July 2018), 74–83. DOI:https://doi.org/10.1145/3213770

9. D. Kondo, B. Javadi, A. Iosup, D. H. J. Epema, The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems, CCGRID 2010: 398-407.

10. Laurens Versluis, Roland Mathá, Sacheendra Talluri, Tim Hegeman, Radu Prodan, Ewa Deelman, Alexandru Iosup (2020) The Workflow Trace Archive: Open-Access Data From Public and Private Computing Infrastructures. IEEE Trans. Parallel Distrib. Syst. 31(9): 2170-2184.

11. Alexandru Uta, Kristian Laursen, Alexandru Iosup, Paul Melis, Damian Podareanu, and ValeriuCodreanu (2020) Beneath the SURFace: An MRI-like View into the Life of a 21st Century Datacenter USENIX ;login. Aug 2020 issue. First FAIR dataset published through Zenodo in 2020. [Data set] http://doi.org/10.5281/zenodo.3878143.

12. Sonja Bezjak, April Clyburne-Sherin, Philipp Conzett, Pedro Fernandes, Edit Görögh, Kerstin Helbig, Bianca Kramer, Ignasi Labastida, Kyle Niemeyer, Fotis Psomopoulos, Tony Ross-Hellauer, René Schneider, Jon Tennant, Ellen Verbakel, Helene Brinken, Lambert Heller, Open Science Training Handbook , Zenodo, DOI: https://doi.org/10.5281/zenodo.1212496, April, 2018.

13. Wilkinson, M., Dumontier, M., Aalbersberg, I. et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. Nature Scientific Data, 3.10.1038/sdata.2016.18.

14. Emery D. Berger, Stephen M. Blackburn, Matthias Hauswirth, Michael W. Hicks: A Checklist Manifesto for Empirical Evaluation: A Preemptive Strike Against a Replication Crisis in Computer Science (2019), https://blog.sigplan.org/2019/08/28/a-checklist-manifesto-for-empirical-evaluation-a-preemptive-strike-against-a-replication-crisis-in-computer-science/, 2019.

15. Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, Alexandru Iosup, Is Big Data Performance Reproducible in Modern Cloud Networks?, NSDI 2020.

2.5 Thesis Contributions

Make a specific list of thesis contributions. There can be multiple flavors of contributions that your thesis does. Make sure to articulate and explain them cleanly. For example:

- Conceptual contribution: here you might have surveyed a field, and categorized knowledge in a new taxonomy, or did a new design of a system, or use an idea from one domain to another, or introduced a new idea or concept, etc.
- Experimental contribution: Here you designed an experiment, evaluated a system, presented experimental guidelines, compared two or more systems in particular aspects by running experiments with them, etc.
- Artifact and dataset contributions: you developed a piece of software which is open sourced and can be used by others, or you collected a valuable dataset (e.g., trace archives (https://wta.atlarge-research.com/traces.html), file system traces (http://iotta.snia.org/traces), I/O traces (https://github.com/Beaconsys/Beacon), cloud reproducibility data set (https://zenodo.org/record/3576604), etc.) that can be used by you and others to conduct further research. The data set should follow the best FAIR data set practices. See "The FAIR Guiding Principles for scientific data management and stewardship" reference above.
- (optional) Knowledge or artifact dissemination: If already during your thesis work you have contributed to open source projects, gave talks and presentations at various venues regarding your work, published a paper (workshop or conference) then you should include such activities here explicitly. These activities help to disseminate your research findings to a wider audience.

2.6 Plagiarism Declaration

I confirm that this thesis work is my own work, is not copied from any other source (person, Internet, or machine), and has not been submitted elsewhere for assessment.

To understand more about plagiarism policy at VU Amsterdam, see https://www.vu.nl/en/about-vu-amsterdam/academic-integrity/index.aspx and https://sites.google.com/vu.nl/academic-integrity-vu/academic-integrity.

2.7 Thesis Structure

If you like to present what the upcoming sections and subsections are going to present, then you can summarize them here. For example you can say that in the next chapter you are going to present detailed information about the concept of "whatever", which is necessary to understand before presenting the design in chapter 3. Prepare your reader regarding what to do.

For more creativity, you can also visualize it, show potential multiple reading paths for readers with different technical expertise.

3 Background

Start each section or chapter on a new page.

Provide necessary background information, concepts, and ideas which are necessary to understand your thesis work. Use examples. Use figures. Use code snippets. Take your time and explain things. Often when we spend too much time with a system we forget what we did not know when we started the work. Keep track of your activities during your thesis work and everything that you had to learn, a new language, a new framework, a new compiler, a new software package design, a new concept all should be explained here cleanly.

How detailed your information should be? Whatever is necessary to understand your contribution. Often a single topic can be really large. For example, if you talk about distributed transactions then the background knowledge can be about providing ACID guarantees, or performance, or concurrency management, or membership management, or consensus, etc. Based on what topic your thesis is tacking, you should only focus on selectively introducing those ideas. You can leave a broader reference saying that *More details about the distributed transaction design can be found in the survey of CoolPeople et al. from ACM CSUR 2020 [x]*. While explaining technical concepts you can slowly prepare the reader about the next chapter and what is about to come.

Take your time to think in which order you should be presenting background information. Do not make a laundry list of random topics. Write them coherently.

At the end of your background chapter, a knowledgeable person or expert in your field should understand your problem completely. It is always better to spell it out clearly. For example, *So far in this chapter we have presented background information about how serverless and container technology work, what are the big container repositories. In Section X.Y we have further elaborated challenges associated with the scalability, complexity and deployment. In the next section, we will present MyCoolSystem that proposes combining light-weight language virtualization with Lambda to deliver high performance, cold booting of Lambdas.*

4 Design or Architecture

Change the heading of the section to make it more specific to match your thesis work. Avoid putting generic heading and titles. Find a unique and search engine index-able name for your project. A proper noun. For example, instead of just saying "Design", put "Design of Anubis: a Fault-Tolerant Extendable File System".

Design or architecture is a high-level concept and idea that you have developed or put together to solve the problem. These are kind of details which are important to re-do your design or project completely from scratch by others in a different setting or system than what you have used. This is the **"transferable knowledge"**. Use the appropriate terminology. Do not use colloquial lingo from the web to describe concepts. If you are unsure, check and read other research papers and pay attention to the language used. Whenever in doubt, ask us.

Important: Do not write frivolous claims. Often in thesis I read "Idea X or design Y is simple" or "It is considered complex" or "everyone uses it". Please do not do this. Every such statement in your thesis should be backed up by a reference. If not, then it is risky to write such claims and you should avoid it. These claims are needed when you have to justify your design or architecture choice, why did you choose one style over other. In a scientific literature, why and justification is very important. Try to reason about your own thinking and working process and write it when describing a design. As usual, if not sure how to justify your choices, ask us.

Can I copy images from the Internet?

First, do not do it. It is often easy to redraw the image by yourself, thus side stepping the image copying issue. But if you must (very discouraged), then you must clearly attribute that picture as "copied from [...]". Make sure that the image is not copyrighted. Also, this is an exception not a rule. You need to make a very strong case for why you want to copy such an image instead of redrawing it.

5 Implementation

Change the heading of the section to make it more specific to match your thesis work.

Implementation section presents how you realize your design from the previous chapter. Here you can use information and challenges regarding what language, system, runtime, compiler, etc. you used. What was challenging about it. Did you use threads, if yes how did you manage them. What kind of I/O API did you use? All these "your" implementation specific details that might be helpful to understand your Github artifact.

Important: Implementation section is not an accompanying documentation to your Github repo.

Avoid using variable names from the code to explain concepts. A simple refactoring of your code repo and this whole chapter becomes useless. Hence, instead of saying "*struct iommu_maps does a callback with void *cb_func pointer to invalidate the struct inode_fs mappings*", say that "*An object is allocated to keep track of IOMMU mappings of file system, which are later invalidated using a callback mechanism*".

You can show code snippets to make your point clear(er). But explain the code properly. Use line numbers in your code illustrations (see the latex package `listings`, and https://www.overleaf.com/learn/latex/code_listing) and say what is happening at what line number. Do not assume that a reader will glance at the code and grasp immediately what you wanted to show. This rule also goes for (i) Graphs; (ii) Equations; and (iii) code on the slides and in the paper. Take your time and explain these 3 things very slowly and carefully. Otherwise they are bound to create more confusion than clear them.

The idea of this chapter is not only to explain in sufficient detail what is required for a reader to leverage and extend the artifact available (useful for **"transferable artifact"**), but also to highlight the implementation complexity and challenges. Systems thesis often requires building a large amount of code before you can show the smallest amount of progress. Hence, make sure the challenges associated with this process are clearly reported and written in the thesis.

As a general guideline, start from the big pieces and big picture, and iteratively drill down (subsection by subsection) into low level details. At the end (or sometime in the start) you can show how all the pieces come together perhaps by means of a *walk through* of a high-level operation. Do not jump up and down in the stack within a single paragraph when explaining concepts.

6 Evaluation

What is the goal of the evaluation section? To generate quantitative data to back up your claims in the thesis of having built a fast/efficient/scalable/better system. Hence, your evaluation section should reflect that. Often I see an evaluation section which is written from a point of view that "we ran the system, and fiddle around with 4 configuration parameters to generate 16 combinations and here are the results". Do not do this.

Setup expectations for the evaluation up front. What you are about to do, how we are doing to do it, and what we have found.

Always include a summary up front

Always include a summary of key findings in a paragraph such as:

Example of Evaluation Summary

The fictitious evaluation of MyCoolSystem is focused around three dimensions (performance, better failure recovery (negative results), and trade-offs):

1. *Does MyCoolSystem deliver lower latencies and higher bandwidths than NotSoCoolSystem?* We evaluate this question in Section X.Y. Our results on a cluster of 10 servers demonstrate that MyCoolSystem delivers 10.1% lower latencies and 50.6% higher bandwidth than the state of the art system NotSoCoolSystem. Key performance benefits from the design decision and implementation optimizations discussed in previous Sections A and B.

2. *Does MyCoolSystem deliver better failure recovery?* We perform failure analysis in Section a.b. Our experiments with the one and two system crashes (fail-stop) show that MyCoolSystem can detect and recover from failures in $39.5msecs$, which is $2.42x$ slower than the state of the art system. Our analysis shows that this slowdown is due to the slow storage devices we are using. We will discuss in our future work (section x.y) potential ideas to optimize the recovery time.

3. *What are trade-offs of using MyCoolSystem?* Our analysis in section $x.y$ shows that MyCoolSystem is the best choice when the performance of the system is bounded by the network performance. Otherwise, MyCoolSystem can incur a performance overhead of up to 34.87%. The overhead is proportional to the number of clients used.

6.1 Reporting Negative Results

Science is about being curious, exploring, and understanding knowledge. By the virtue of this process, we can not guarantee success every time we start to work on

a problem. If the result was known up front then what is the value of your work? If all experiments were always successful, we would be living in a very different world.

We understand that due to multiple reasons a thesis might not work out and you will not get expected positive results. Report these results truthfully, and focus your attention to explain why it has happened. Why did we think at the start that this thesis should result in a positive evaluation of the system. What went wrong, did the development become complex, hardware was not adequate, other technical reasons? Try to speculate how it could have worked under what conditions.

Negative result is an equal and valid contribution from your thesis, given than the initial premise of the experiment was sound and justified.

6.2 Experimental Setup

Have a clean subsection where you can provide detailed setup information that might be necessary to reproduce your experiments. These details might include hardware and software configurations, operating systems, what features are enabled or compiled in, (if applicable) which datasets and traces are used in the evaluation.

What are Micro and Macro benchmarks

A micro benchmark is a benchmark that selectively exercises and benchmark a particular code path in your program. For example, if you are building a file system you may want to micro benchmark a specific read or write call. Microbenchmarks are typically done in depth, dissecting the performance and overheads in detail, and explaining what is happening in the system when processing a particular request. In contrast, macro benchmarks take a broader view (horizontal view) and try to answer the question how your particular system behaves as a whole, often when integrated with other applications and frameworks. Coming back to the file system example, in macro benchmarks you may want to benchmark how file system benchmarks like cloning and building the Linux source tree, a file or email server, big data workloads like Databases, etc. benefit or do not benefit from your file system. In macro benchmarks, applications might be doing a lot of different things including read and write calls. Based on the type of work you are doing, you may or may not have to do one of these two types of benchmarking.

Designing an Experiment

Experiment design is a very important scientific skill. It is like writing good code, or a good thesis. How can you design an experiment that shows or measure what you want to measure? How can you be sure if you measured the right thing? I often ask students, what if I do not believe the experiment that you are showing me, can you convince me otherwise that you are doing a correct measurement. All these questions are very important and take time to think, and design an experiment. Doing a good experiment would also involve doing coding to generate and plot the right set of numbers. Try to automate the whole process of doing experiments as much as possible.

In designing an experiment, try to vary one parameter at a time to study its impact. Do not choose arbitrary values such as looping for 10 times, why not 5 or 100 times? Always try to justify all possible value selections. Often these values are either conventions, values that you might have seen in other papers, or just default that you choose to leave without changing.

Understand Time

Be careful when you measure time. Make sure you understand what is the overhead of time measurement. https://linux.die.net/man/7/time. Try to amortize the overhead of time measurements. Also consider what is the event that you are trying to measure, does that event happen in a second, millisecond, microsecond, or nanosecond granularity. Make sure that your experiment runs at-least a couple of orders of magnitude longer than the precise event that you are trying to measure. For example, if you are measuring a page fault that takes 10 microseconds, then design an experiment that generates one million faults, thus running your experiments in 10s of seconds. Do not blindly calculate the average of values. Report appropriate box-and-whiskers plot with min, max, median, 95 and 99 percentile values.

Do Quick and Dirty Calculations

Always do back-of-the-envelope calculations quickly. Try to guesstimate what you are about to measure, how long it should take, how fast it can be, etc. Then verify your understanding and intuition from the experimental results you get. For example, if you measure that on the DAS-5 you can perform 10 million RPC operations per second for 1kB bytes request and response, then I know your measurements are wrong. These measurements suggest that DAS-5 can do 10,000,000 x 1024 bytes = 81.92 Gbps. However, DAS-5 uses FDR InfiniBand network, which is limited to 56 Gbps. Hence, you must have measured something wrong here. Such quick and short calculations are very important to verify the sanity of your experiments. Try to think creatively how else can you check and measure such things.

Designing experiments and evaluating systems is a tedious and complicated process. Only by being meticulous and comprehensive we can avoid common mistakes and pitfalls.

Reporting on Experiments

For each experiment that you conduct and report, write cleanly in the thesis:

- What you are about to do? For example, *in this experiment we are going to compare the run-time overheads of system X with our system Y.*
- Why such an experiment is relevant. *One of the aims of the thesis work is to reduce the runtime overheads, hence, this experiment will let us compare directly the overheads in presence of feature A or not.*
- How did we measure this? *We deployed system X and Y, and calculated the number of operations completed per second. Each operation takes 3*

different steps, which we measured individually and sum up the time taken. All experiments are run one million times and we report 95 percentile. The complete measurement data-set is available in Appendix A.

- Discuss graphs and results in detail. Take your time to explain them properly. Point out data points in the graphs which are visible, and others that you know from the raw data points, but which might not be so visible in the graph. Any further information that you did not plot.
- What are limitations of this experiment? For example, *We measured the impact of feature "A" indirectly as it was not possible to modify the source code. Or we could not get software "X" to work under desired conditions. Or our hardware is old and the new generation of hardware has a fix for this bug.*

Often the shape of the graph does not show an expected figure. In that case, you can state clearly that there is an anomaly and we do not know what caused it. If you have an idea then you can write, e.g., *As shown in the graph, for request sizes between 8-16MB the time taken to process requests increases significantly (by 80.1%). We suspect that this is due to increased LLC cache misses, however, we have not verified it.*

Do not round numbers such as `34.73` to 35 or 34. Always report accurately to the 2 decimal points. In systems research, I rarely see full rounded numbers like 200% or 10 times.

Plotting Graphs

Use a proper tool to plot high-quality, proportionally front graphs that can be read properly without zooming in. Use vector graphics, not bitmap. No jpeg, jpg, or png please. Often students start with these formats as a placeholder, then in the end before the thesis submission it is too late to re-plot the graphs. Please take time to learn and set up graph plotting tools.

One message or finding per graph: Each graph should have only one key message it should convey (max two, using two y axes). Do not try to cramp too many data points in a single graph. You can easily split them into multiple figures. Strive for simplicity, and elegance!

Explain your graphs: Always explain how a reader should read your graphs. Do not say as shown in the figure we build a cool system. No. For every graph that you include in the thesis you should explain what it is. Always fill out 4 sentences automatically (i) what is it showing; (ii) what is on the x axis (any special note like log axis);; (iii) what is on the y axis (any special note like log axis); (iv) what should a reader read from the graph, what is the expected shape, lower or higher is better, or any other special mention. For example:

Example of a graph explanation in a figure

Figure X shows (or plots) the number of operations completed with respect to the number of cores in the system. The x axis shows the number of active CPU cores (skipping the special core 0). The y axis (log scale) represents performance gains in Million of operations per second (MIOPS), higher is better. The graph shows that as we increase the number of active cores from 2 to 16, the number of operations completed is increased from 20.12 MIOPS to 104.87 MIOPS, an increase of 421.22%. The system suffers a performance drop of 5.52% between core scaling of 11 and 12 CPU cores (not visible in the graph), which is caused by CPU voltage-frequency scaling mechanism of Intel CPUs.

Pick sensible axis ticks: Pick sensible values for which you want to do experiment and plot data. For example, for an experiment with file sizes, do not pick file sizes of 1kB, 2kB, 10kB, 50kB, 100kB, and 1MB — a seemingly random sequence of sizes. Pick a clear trend (log or linear values) like a log sequence of 1kB, 2kB, 4kB, 8kB, ...1MB (2x or 4x jumps) or 1, 10, 100, 1000 (base 10 log). Do not mix and match random and arbitrary sequences.

Do not plot break graph axes arbitrarily. Do not plot the y axis randomly from an arbitrary value. Always use 0 (for a linear plot), or 1 (for a log plot).

6.3 Limitations and/or Threat to Validity

What are you missing from your evaluation? Under what circumstances results from your evaluation will change substantially? By how much (any reasonable estimate)? What features are missing from the implementation that you did not get to evaluate? Any specific experiment that you could not run due to any specific issue in the setup? Did you use old hardware, and how your results will change if you are to use the new hardware? Did you use an emulator and/or simulator, do you suspect that your results might change if you were evaluating in a real world? Or some part of the setup or optimization about which you are not completely sure that you did 100% to check all possibilities.

6.4 Summary or Discussion on Evaluation

Your evaluation will consist of multiple subsections. At the end of each subsection you should clearly summarize what is the conclusion from that experiment. At the end of your evaluation chapter you should have a final subsection which is the summary of the whole evaluation. Here you should clearly recap what the evaluation section has shown (quantitatively!). Use as many numbers and data points possible. Connect the results from the evaluation back to your system architecture, design choices, performance claims, or research questions that you made at the start of the thesis. Have an open discussion about the evaluation of what has worked, and what has not worked.

7 Related Work

It is often a misconception that you need to defend against the related work. Related work is your friend. Show your evaluator that you are aware of the state of the art in your field of research. Think critically, which small design choices and pieces remind you of which paper. What is your original contribution and what you can claim from others. More comprehensive your related work is, the more I will believe your contributions.

Where can I find related work?

At the start of your thesis I will give you an initial set of papers and conference names. Read those papers. Pay attention to their related work, and read those. Synthesize the common keywords and terminology that these papers use, and then search those on Google Scholar, or any other academic research engine. See which conferences those papers are published in, and then go check previous iterations of those conferences. Pay attention to the research groups and universities who write those papers, and then go check their other works on their homepage. Even after all of this you cannot find any other paper, come talk to me, but then you have to show me what you have done so far.

How many references are enough?

There is no right answer here. For a well established filed, you might be looking papers in 50-100s. For a new field, there might only be a handful of papers. Try to identify the impactful papers and try to include them. However, in any circumstances including one or two papers is never good. In that case try to expand what constitutes related work and show that you are aware of a new established field and other nearby areas from which the new field gets ideas. You can be creative with this. For example if you do not find enough related work on programmable storage, then you can include programmable network, languages, compiler work in your related work.

What happens if I miss a paper?

No need to panic. In broad and fast moving research areas of Computer Science, there will be non-zero probability that we might have missed a paper. Often it is a simple process just to add the missed paper in the discussion without much restructuring. Though it is always preferred to avoid such situations, especially if the missed paper is really an important one.

For more information regarding how to do a survey in a new filed, please see my advice on how to do a literature study https://animeshtrivedi.github.io/lit-study/.

8 (Optional) Lesson Learned

Based on the nature of the work, it might be applicable to the key recommendations from your thesis work, or your experience in a separate chapter. These lessons can be scientific like particular design choices (e.g., a non-blocking asynchronous I/O call with a preemptive schedule is hard to get right, or correct consensus implementation is hard to get right in the first place). It could be more mundane lessons like, code proper code management, always measure a particular variable, or do not trust documentation, etc. But back these up how arrived at such lessons.

A very good example of lesson learned are these papers (I am sure there more examples out there):

- Andrew S. Tanenbaum. 2016. Lessons learned from 30 years of MINIX. Commun. ACM 59, 3 (March 2016), 70–78.
- Gernot Heiser and Kevin Elphinstone. 2016. L4 Microkernels: The Lessons from 20 Years of Research and Deployment. ACM Trans. Comput. Syst. 34, 1, Article 1 (April 2016), 29 pages.

9 Conclusion

Read your Abstract and Introduction sections and then write your Conclusion. A reader should be able to read these three sections in one go and get the gist, problem, research findings, and direction of your work.

9.1 Answering Research Questions

Have your research questions clearly answered explicitly here. An acceptable answer is either qualitative or quantitative. Qualitative answers are in which you provide a detailed analysis of an area. For example, *What are the important parameters to consider when designing a storage system?* You can answer various parameters that you have identified, why they are important, and what the literature says about them. Quantitative answers are backed up by your evaluation section. For example, *What is the performance overhead of file systems for small files?* You can answer that based on multi-media file access patterns (large sequential scans), the overhead is between 5.82-104.24%. Provide more detailed steps that you took to answer the question.

One important aspect of answering research questions is to provide hints and directions in which these results can be used by the reader. For example, you can say that based on our findings, we recommend to optimize a certain operation, or measure a setup before deploying, or synthesize the value of a particular parameter. What is the transferable knowledge here? Think and write about that.

9.2 Limitation and Future Work

Clearly identify the limitation of the current work, e.g., incomplete implementation, lack of measurements, inconclusive experiments, additional features, lack of time for a particular investigation, etc.

What kind of limitations can you discuss?

Here are some ideas

- Some experiments which you could not manage to run?
- Use of old hardware, and how your results will change if you are to use the new hardware?
- Use of emulator/simulator, if you suspect that your results might change if you were evaluating in a real world.
- Missing features or ideas that you have not implemented, or evaluated?

10 References

The final section with references. The thesis template will generate the links in references for you. Please make sure that all references are consistent in formatting, and there are no broken links. Please do not leave for me to check and correct broken links for you. It is very annoying. Be vigilant and thorough.

Appendix A: Reproducibility (Mandatory)

This is a mandatory appendix. You must include this in your thesis. This subsection can be designed around the ACM guidelines for experiment reproducibility at https://www.acm.org/publications/policies/artifact-review-badging. ACM offers badges in three classes: Artifacts Evaluated, Artifacts Available and Results Validated. Understand what they mean and provide sufficient information for someone completely new to take your code and run it on a new machine.

The least I expect is a running code. Hence, you should clearly describe how to download, setup and run the code on a new machine. How to run a representative experiment that can verify if the software is working. And lastly, how to reproduce one of the key experiments from your thesis and reproduce the exact data and graphs. For more specific guidelines regarding what to include in this subsection, please see https://ctuning.org/ae/checklist.html and use the latex template provided here https://github.com/ctuning/ck-artifact-evaluation/blob/master/wfe/artifact-evaluation/templates/ae.tex. Fill in the relevant sections.

I prefer that you make your code available on Github. A part of the documentation that you will write here can also be used in the top-level README.md file for your Github repo. Make sure no broken or private repo links are used. If you have used private repo to develop the code, then make sure to make it public before your thesis is completed.

For data repositories you can consider https://zenodo.org/. For example, see this dataset from Cloud performance reproducibility experiments https://zenodo.org/record/3576604.

Appendix B: Self Reflection

Explain in this section how this research benefited yourself. How did this project develop you? How did it develop your career? Also include here quantitative elements: break down the time spent on this project by type of activity / element of the project.

Appendix C: Additional Experiments

If you have additional experiments to report, you can include them here.