Advanced Topics in Distributed Systems

Paper writing

Guillaume Pierre

Vrije Universiteit

Fall 2010
http://www.cs.vu.nl/~gpierre/courses/atds/

vrije Universiteit amsterdam

Table of Contents

Disclaimer

It is impossible to teach you everything about paper writing and/or presentations in one class!

- You will have to practice again and again and again and again and again...
- But there are a few classical techniques and pitfalls that you should know about

Table of Contents

Why is writing quality important?

"The fundamental nature of scientific discourse is not the mere presentation of information and thought, but rather its actual communication. It does not matter how pleased an author might be to have converted all the right data into sentences and paragraphs; it matters only whether a large majority of the reading audience accurately perceives what the author has in mind."
George D. Gopen and Judith A. Swan, "The Science of Scientific Writing."

Important axiom

Axiom
The reader is lazy!

- Assume that **the reader will make minimal efforts** to understand you
- It is **your task** to make sure the reader cannot possibly misunderstand you
  - If a reasonably competent reader does not understand you, then **you are the one to blame** (not the reader)
- Note: this requires a lot of work :-(

When should you write a paper?

1. You have a well-defined problem
   - Finding the right problem to solve is half of the work!
   - Do not duplicate existing work
   - Do not work on ridiculous problems
2. You have **one** good idea
   - The whole paper should be targetted towards defending this single idea
   - **Two ideas** ⇒ **two papers** (you cannot chase two rabbits at the same time)
3. You can demonstrate how good your idea is
   - You fully understand it
   - You have some form of evaluation (mathematical proof, modeling, simulations, real-world experiments, etc.)

Special case: position papers

- A position paper is meant to allow one to **defend a position**
  - An opinion, a new (untested) idea, etc.
- Bold ideas and provocative statements are welcome!
  - *"Resolved: Publish no more papers on Web cache replacement policies"* (panel title at the Workshop on Web Caching and Content Distribution, 2001)
- You do not need a full proof/evaluation
  - You must only **build an argumentation**: why do you believe in this? Why would anyone agree with you?
  - You will need to think carefully about the way to present your ideas (more on this later)

Good position papers

- Good position papers contain:
  - One clear-cut/bold/controversial **claim**
  - **Some** reasonable argumentation
- **Write your position in the title of the paper:**
  - *"Gossip-based algorithms can be great for Vehicular Ad-Hoc Networks"*
  - *"Vigilante Does Not Stop Large-Scale Malware"*
  - *"Virtualization is the natural evolution of Operating Systems"*
  - *"Debugging Distributed Systems (a new idea)"*
- **We can agree to disagree!**
  - You can get a very good grade even if I disagree with your claim
  - Provided that you present a decent argumentation

Bad position papers

- Bad position papers contain:
  - No clear position
  - Vague argumentation
  - Lots of handwaving
- Example: surveys hidden into a position paper
  - Title: *"A Survey on Topic X"*
  - Section 1 (Introduction): *"Topic X had attracted a lot of attention recently, let's study the proposed solutions."*
  - Section 2: *a summary of paper 1*
  - Section 3: *a summary of paper 2*
  - Section 4: *a summary of paper 3*
  - Section 5 (Conclusion): *"Different solutions have been proposed on topic X. However, each one has some strengths and weakness so more research is needed on this topic."*

Such papers will receive *grade* ≤ 5 **!**

New this year

Email me your position before writing your position paper.

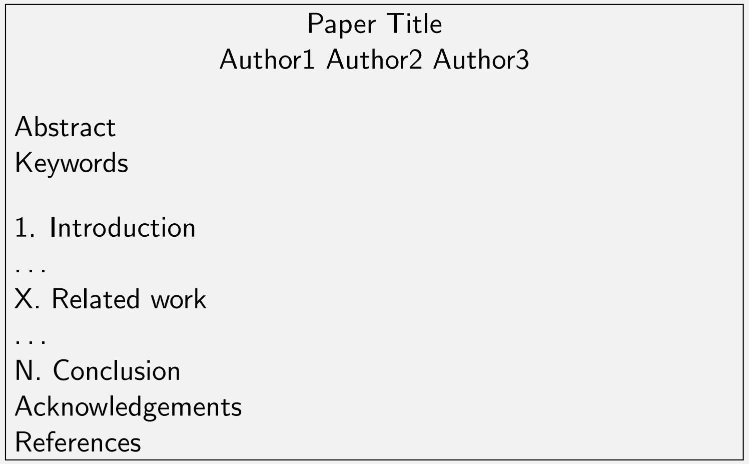
- No argumentation needed, just your claim. **Maximum size: 20 words.**
- I will tell you if I think this is a suitable position

Table of Contents

Paper Structure

- Every paper is different (of course)
  - Different content
  - Different audience
  - Different size limits
- But somehow, (nearly) all papers are structured the same
  - Let's study the commonalities

General Paper structure


Paper Title
Author1 Author2 Author3

Abstract
Keywords

1. Introduction
. . .
X. Related work
. . .
N. Conclusion
Acknowledgements
References

Title

- What is your paper about?
  - Nobody cares what you did: "*A New Way to Maintain Links In Between Nodes Such That They Can Reach Each Other In Log(N) Hops*"
  - Better focus on the **scientific significance** of your work: "*A Scalable Content-Addressable Network*"
- **Finding a good title is not easy at all**
  - It forces you to think hard
  - "Why would anyone care about my work?"
  - "What should people remember about my work?"
  - "What is the take-home message?"

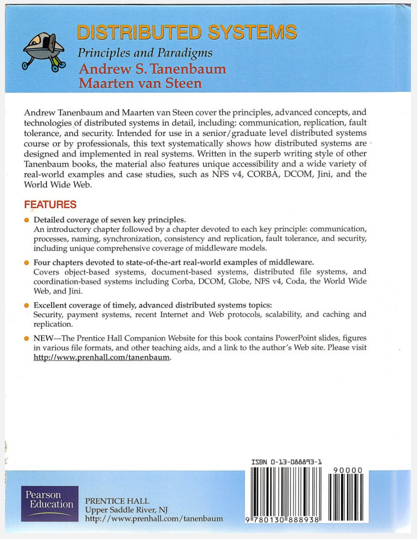
Authors

- Everybody who directly contributed to the paper should be listed as author:
  - The colleague who came up with the initial idea, the researcher who carried the research, the student who helped running the experiments, the supervisor(s)
  - The friend who gave useful comments but did not directly produce stuff does not show up as an author (you can acknowledge his/her help in a special section after the conclusion)
- Authors are shown in decreasing order of contribution
  - First those who did all the hard work
  - Then those who helped
  - Then those who supervised

Table of Contents

What is an abstract?

- Many people confuse abstract and introduction!
- The abstract does not have a section number
- The abstract is a **summary** of your paper
  - It is used for readers to quickly decide if your paper is worth reading or not
  - If you print the article as a book, then **the abstract is placed on the back cover**
  - Size: about 200-300 words



DISTRIBUTED SYSTEMS

Principles and Paradigms

Andrew S. Tanenbaum

Maarten van Steen

Andrew Tanenbaum and Maarten van Steen cover the principles, advanced concepts, and technologies of distributed systems in detail, including: communication, replication, fault tolerance, and security. Intended for use in a senior/graduate level distributed systems course or by professionals, this text systematically shows how distributed systems are designed and implemented in real systems. Written in the superb writing style of other Tanenbaum books, the material also features unique accessibility and a wide variety of real-world examples and case studies, such as NFS v4, CORBA, DCOM, Jini, and the World Wide Web.

FEATURES

- **Detailed coverage of seven key principles.** An introductory chapter followed by a chapter devoted to each key principle: communication, processes, naming, synchronization, consistency and replication, fault tolerance, and security, including unique comprehensive coverage of middleware models.
- **Four chapters devoted to state-of-the-art real-world examples of middleware.** Covers object-based systems, document-based systems, distributed file systems, and coordination-based systems including Corba, DCOM, Globe, NFS v4, Coda, the World Wide Web, and Jini.
- **Excellent coverage of timely, advanced distributed systems topics:** Security, payment systems, recent Internet and Web protocols, scalability, and caching and replication.
- NEW—The Prentice Hall Companion Website for this book contains PowerPoint slides, figures in various file formats, and other teaching aids, and a link to the author's Web site. Please visit http://www.prenhall.com/tanenbaum.

Pearson Education

PRENTICE HALL
Upper Saddle River, NJ
http://www.prenhall.com/tanenbaum

ISBN 0-13-088893-1

90000

9 780130 888938

Contents of an Abstract

- What is the problem you are trying to solve? (quick and to the point)
  - Note: it does **not** look stupid to explain what the problem is!
- What do you propose to solve the problem?
- Announce good performance results (if you have some)

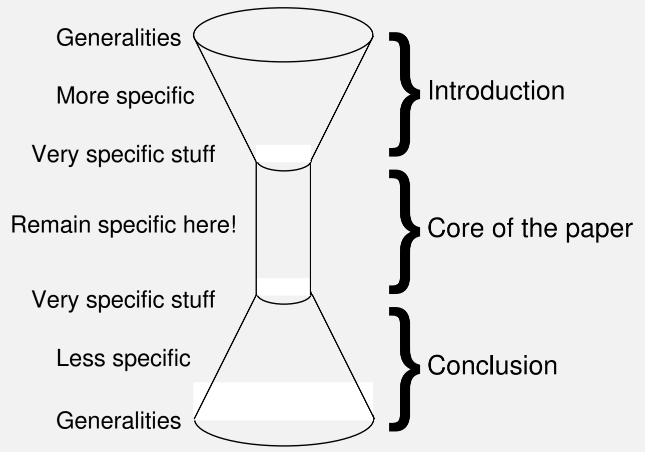
Abstract
Problem: A fundamental problem that confronts peer-to-peer applications is to efficiently locate the node that stores a particular data item. **Solution:** This paper presents Chord, a distributed lookup protocol that addresses this problem. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. Chord adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. **Results:** Results from theoretical analysis, simulations, and experiments show that Chord is scalable, with communication cost and the state maintained by each node scaling logarithmically with the number of Chord nodes.

Table of Contents

What is an introduction?

- This is the "brainwashing" part of the paper
  - Input:
    - A reader in any possible state of mind (tired, bored, has no clue what you are talking about, etc.)
    - **The reader has not read the abstract!**
  - Output:
    - The reader knows what problem you are trying to solve
    - (S)he is convinced that this problem is worth solving and wants to know more
    - (S)he knows how you plan to solve the problem
- The introduction of a paper is by far **the most difficult part to write**
  - Fortunately, 99% of introductions follow the same structure
  - Nothing extraordinary, but it just works

The hourglass model



Structure of an introduction

1. **Establish context: what are the surroundings of this work?**
   - *"Peer-to-peer file sharing is getting increasingly popular..."*
   - Note: you do not need to be very original here ;-)
2. **What is the problem you are trying to solve?**
   - "*Many applications need efficient mechanisms to search for specific content among large numbers of machines*"
   - Note: many people forget to explain what the problem is. Wrong!
3. **Why should one care? Why is the problem difficult?**
   - "*Traditional flooding mechanisms do not scale. On the other hand it is hard to maintain a centralized index up-to-date*"
   - Note: this part comes close to a tiny related work section
4. **What is the main idea to solve the problem?**
   - "*We give an ID to each node and data item; a data item is under the responsibility of the node with the closest ID*"
   - Note: you must give just enough details so that the reader can start guessing what your solution looks like

(to be continued...)

Structure of an introduction (cont.)

5. **Explain what is great about your solution**
   - "*This allows one to very simply build a whole range of cool applications: you just need to use the route() function to reach the right node*"
   - Note: don't be afraid to explain what is great about your work! If you don't do it, nobody else will do it for you
6. (optional) **Announce performance results**
   - "*We show using simulations that our system scales to millions of nodes; we deployed it on PlanetLab and proved that it handles churn very well*"
   - Note: Tell the reader that this is for real (so many papers promise great stuff until you realize 10 pages later that they never really built the system)
7. **"This paper is structured as follows..."**
   - Note: this part is necessary so that the reader does not get lost in the paper. "Don't worry, we *will* discuss implementation details but you must first understand how the abstract algorithm works"

Example introduction

SplitStream: High-Bandwidth Multicast in Cooperative Environments

Context: End-system or application-level multicast [8, 16, 22, 19, 40, 32, 13, 6, 23] has become an attractive alternative to IP multicast. Instead of relying on a multicast infrastructure in the network (which is not widely available), the participating hosts route and distribute multicast messages using only unicast network services. **Problem:** In this paper, we are particularly concerned with application-level multicast in peer-to-peer (p2p) or cooperative environments where peers contribute resources in exchange for using the service.

(continued. . .)

Example introduction

The problem is hard: Unfortunately, conventional tree-based multicast is inherently not well matched to a cooperative environment. The reason is that in any multicast tree, the burden of duplicating and forwarding multicast traffic is carried by the small subset of the peers that are interior nodes in the tree. The majority of peers are leaf nodes and contribute no resources. This conflicts with the expectation that all peers should share the forwarding load. The problem is further aggravated in high-bandwidth applications, like video or bulk file distribution, where many peers may not have the capacity and availability required of an interior node in a conventional multicast tree. SplitStream addresses these problems by enabling efficient cooperative distribution of high-bandwidth content in a peer-to-peer system.

(continued. . .)

Example introduction

Key idea: The key idea in SplitStream is to split the content into k stripes and to multicast each stripe using a separate tree. Peers join as many trees as there are stripes they wish to receive and they specify an upper bound on the number of stripes that they are willing to forward. The challenge is to construct this forest of multicast trees such that an interior node in one tree is a leaf node in all the remaining trees and the bandwidth constraints specified by the nodes are satisfied. This ensures that the forwarding load can be spread across all participating peers. For example, if all nodes wish to receive k stripes and they are willing to forward k stripes, SplitStream will construct a forest such that the forwarding load is evenly balanced across all nodes while achieving low delay and link stress across the system.

(continued. . .)

Example introduction

Announce great properties: Striping across multiple trees also increases the resilience to node failures. SplitStream offers improved robustness to node failure and sudden node departures like other systems that exploit path diversity in overlays [4, 35, 3, 30]. SplitStream ensures that the vast majority of nodes are interior nodes in only one tree. Therefore, the failure of a single node causes the temporary loss of at most one of the stripes (on average). With appropriate data encodings, applications can mask or mitigate the effects of node failures even while the affected tree is being repaired. For example, applications can use erasure coding of bulk data [9] or multiple description coding (MDC) of streaming media [27, 4, 5, 30].
The key challenge in the design of SplitStream is to construct a forest of multicast trees that distributes the forwarding load subject to the bandwidth constraints of the participating nodes in a decentralized, scalable, efficient and self-organizing manner. SplitStream relies on a structured peer-to-peer overlay [31, 36, 39, 33] to construct and maintain these trees. **Performance:** We implemented a SplitStream prototype and evaluated its performance. We show experimental results obtained on the PlanetLab [1] Internet testbed and on a large-scale network simulator. The results show that SplitStream achieves these goals.

Example introduction

Structure: The rest of this paper is organized as follows. Section 2 outlines the SplitStream approach in more detail. A brief description of the structured overlay is given in Section 3. We present the design of SplitStream in Section 4. The results of our experimental evaluation are presented in Section 5. Section 6 describes related work and Section 7 concludes.

Table of Contents

What is a related work section?

- Discussing the related work is **imperative**
  - To **tell the reader** that you know what you are talking about
  - To **teach the reader** about previous work that you are going to use
  - To **put your work in perspective** (How does it differentiate from the 12,000 previous papers written on the topic? Why should I read this one?)
- The keyword in "related work" is **related**
  - What is the **relationship** between this previous work and yours?
  - Background, previous work that you want to extend, competitor, etc.

Where to place the related work section?

- The related work section can be Section 2 (after introduction) or $N-1$ (before conclusion)
- Deciding where to place it depends on your type of argumentation:
  - If you **build upon previous work** $\Rightarrow$ Section 2
    - ★ The reader must know the previous work to understand your solution
    - ★ **This is usually the best place**
  - If you **oppose previous work** $\Rightarrow$ Section $N-1$
    - ★ Better to first present your solution, then explain why it is better than its competitors

Structure of a related work section

- Sort related work from the simplest/less relevant to the most complex/most relevant.
- If several works take the same approach, discuss them together!
  - First present the commonalities
    "A number of approaches model BitTorrent using a fluid model [3,4,5]. This allows to derive differential equations on the transfer state of each peer, which can then be resolved numerically."
  - Then present the differences
    "Authors in [5] dropped the assumption that all peer capacities are homogeneous, yet at the cost of a more complex model."
- For each system (or group of systems) you **must** have three parts
  1. Objective description: What does this system do?
  2. Positive qualities: why is this paper relevant to your work? What is interesting in it? What can be reused for your system?
  3. Negative qualities: Why is this work not sufficient? (it addresses a different problem than yours, it is sub-optimal, etc.)

Example related work section

(Neutral description:) Traditional approaches for resource allocation in distributed environments leverage off centralized or hierarchical architectures in which a few servers keep track of all the resources in the network and offer search functionality to the users [1]. **(Qualities:)** While these solutions are well-suitable for clusters or small collections of PCs, **(Drawbacks:)** they exhibit serious scalability issues when the system size increases. Hence, in recent years, researchers put forth a large effort to devise decentralized solutions addressing large-scale scenarios [2].

(continued. . .)

Example related work section

(Neutral description:) The vast majority of these works exploit DHTs to map resources to nodes in order to distribute the search operations across different nodes. Early works [3,4,5] maintain a separate DHT for each attribute: each query is executed in parallel on each overlay networks and results are then intersected. Alternative approaches [6,7], instead, reduce the d-dimensional space to a 1-dimensional space by means of a Space Filling Curve, thus reducing the problem to routing in 1-dimensional space. Finally, more recent works [8,9,10,11], inspired by CAN [12], partition the d-dimensional space into smaller blocks which are assigned to specific nodes which are responsible of all resources falling in that block.

(continued. . .)

Example related work section

(Qualities:) Although these approaches provide high levels of scalability (as opposed to centralized solutions), **(Drawbacks:)** they exhibit two major limitations. First, they significantly suffer from skewed distribution of nodes. In many environments, it is likely that resources are not uniformly spread in the d-dimensional space. Consequently, the nodes that are responsible for the dense areas can experience overload. Some of the aforementioned works address this issue by periodically running an additional protocol to reallocate the zones. This, however, increases the complexity of the protocol and does not provide any strong guarantee. Second, DHT-based approaches require that each node is represented by another node in the system. This can create inconsistency problems between the resource and its representative in the DHT. In case of a resource failure, a failure detector must explicitly update the DHT. Conversely, resources might become unreachable because their representative in the DHT has just disconnected.

(continued. . .)

Example related work section

(What is the relationship:) In our approach, each resource represents itself in the overlay instead of relying on an external entity. This removes the consistency problem: when a node's properties change, or the node fails, the overlay reconfigures itself very quickly through the underlying epidemic dissemination. Also, as shown in Section 5, our protocol balances the query load among nodes, even in the case of non-uniform distribution of nodes in the d-dimensional space.

- Question: guess if this related work section was placed as Section 2 or Section $N - 1$?

Table of Contents

The conclusion serves three purposes

1. Announce the **take-home message**: if the reader remembers **only one thing** about your work, what should it be?
   - E.g., "restructuring data according to query access patterns allows one to obtain significant scalability"
   - No need to re-re-re-explain what you did: "we applied linear regression techniques and found that it works fine"
2. **Draw conclusions** from your work
   - What is the significance of your work?
   - What are the limitations?
   - **What have we learned?**
3. Explain **in what sense will the world be a better place** now that you have solved this problem
   - Most movies to not end just after the good guy killed the bad guy.
   - The good guy still needs to get back home, release his girlfriend and walk with her towards the sunset...

Example conclusion

Take-home message: Most approaches toward scalable hosting of Web applications consider the application code and data structure as constants, and propose middleware layers to improve performance transparently to the application. This paper takes a different stand and demonstrates that major scalability improvements can be gained by allowing one to denormalize an application's data into independent services. While such restructuring introduces extra costs, it considerably simplifies the query access pattern that each service receives, and allows for a much more efficient use of classical scalability techniques. We applied this methodology to three standard benchmark applications and showed that it allows TPC-W, the most challenging of the three, to scale by at least an order of magnitude compared to master-slave database replication. Importantly, data denormalization does not imply any loss in terms of consistency or transactional properties. This aspect makes our approach unique compared to, for example, [18].

(continued...)

Example conclusion

What is the significance of this? In our experience, designing the data denormalization of an application from its original data structure and query templates takes only a few hours. On the other hand, the work required for the actual implementation of the required changes highly depends on the complexity of each data service.
What are the limitations? Data denormalization exploits the fact that an application's queries and transactions usually target few data columns. This, combined with classical database denormalization techniques such as query rewriting and column replication, allows us to cluster the data into disjoint data services. Although this property was verified in all applications that we examined, one cannot exclude the possible existence of applications with sufficient data overlap to prevent any service-oriented denormalization. This may be the case of transaction-intensive applications, whose ACID properties would impose very coarse-grained data clustering. It is a well-known fact that database transactions in a distributed environment imply important performance loss, so one should carefully ponder whether transactions are necessary or not.

(continued...)

Example conclusion

The world will now be a better place: The fact that denormalization is steered by prior knowledge of the application's query templates means that any update in the application code may require to restructure the data to accommodate new query templates. However, the fact that all data services resulting from denormalization have clear semantics makes us believe that extra application features could be implemented without the need to redefine data services and their semantics. One can also imagine to fully automate denormalization such that any necessary change in the data structure could be applied transparently to the application, using a proxy layer to translate the original application query templates into their data service-specific counterparts. We leave such improvements for future work.

Table of Contents

When should you throw in a figure?

- **To present evaluation graphs** (of course)
- But also: **when a figure is more clear than 1000 words**
  - When you write text and explain something, sometimes you realize that you formed a mental picture that you try to describe in writing
  - Plot the picture!

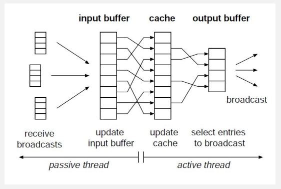


Figure 1: Visual representation

We abstract a framework to describe the core structure of a replication and storage protocol like SharedState. There are three main operations (see Figure 1) that a node needs receive broadcasts input buffer cache output buffer update input buffer update cache select entries to broadcast broadcast passive thread active thread Figure 1. Visual representation. to execute: a) handle incoming items (passive mode), b) update its cache (active mode) and c) select which items to broadcast (active mode). The specific way in which these three events are implemented has a direct impact on the characteristics of the propagation and replication of items.

How to throw in a figure

- **A figure should be self-contained**
- A figure must have:
  - A figure number, a legend and a caption
  - Figures should be placed on top of the page
- The main text must contain:
  - A reference to the figure number
  - An explanation of what the graph/figure shows
  - A (short) discussion about the important features of the graph (what is the reader supposed to notice?)
  - (for performance graphs) A conclusion: what did we learn from this graph?

Example figure

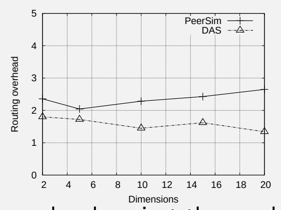
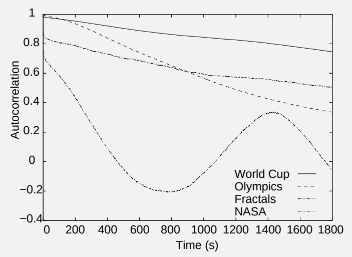


Figure 5: Routing overhead against the number of dimensions.

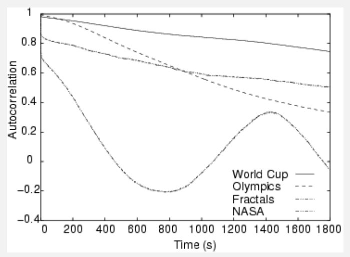
A major difficulty in multidimensional peer-to-peer systems is to be able to handle a large number of dimensions, which in our system correspond to node attributes. **(Figure number:)** Figure 5 charts **(What is plotted:)** the performance of our algorithm when using different numbers of dimensions, in both PeerSim and DAS setups. **(What do we see:)** In the PeerSim experiments, the overhead increases slightly with the number of dimensions, while in the DAS setup it remains roughly constant. These variations, however, remain difficult to interpret, as such low overhead values typically fall within normal statistical error margins. **(Conclusion:)** What is noticeable, however, is that in all cases the routing overhead remains extremely low.

Vector vs. bitmap graphics

- Always generate graphs in a **vector graphics** format
  - And insert vector graphics into your report: EPS, PDF, SVG, etc.
  - Avoid bitmap formats: GIF, JPG, PNG, etc.



Vector



Bitmap

- Both formats show more or less the same on screen, but print this page and you will see the difference. . .
- It matters less when you prepare a presentation (beamers are good at antialiasing)

The first paragraph problem

- What do you write in the first paragraph of a section?

2. Related Work
Here I need to write up something but I don't know what. "In this section we describe the related work" looks really stupid, doesn't it? Help!
2.1 First topic to discuss
. . .

- Writing something there is **mandatory**
- Explain **the reason why** the reader should keep on reading:
  "Our solution is based on a number of techniques that have been previously proposed by X and Y. Let us first discuss them, after what we will turn to our wonderful solution."

Table of Contents

Typical dialog

– Reader: “I didn’t understand your paper until page X because you didn’t tell me information Y. You are lucky I didn’t give up reading before that.”

– Writer: “Come on, it is written there! Look, page X-10 in the middle if this paragraph!”

– Reader: “I see it now that you show me, but I hadn’t noticed it at first read”

– Writer: “But that sentence was clear, what more can I do?”

– Reader: “Your paper is badly organized. Information should be placed at the location where I expect to find it; that is, you must sort ideas in a logical order, and emphasize only the important stuff”

Display the map of the maze

- Your paper probably presents complex ideas
  - They seem simple to you now
  - But it took you X months to figure them out!
  - Would these ideas have been so simple to you X months ago?
- **It is your duty to make sure the reader does not get lost**
  - Regularly remind the reader: what is the point that the paper is trying to make? How did we get into discussing this particular point?
  - In introduction of course
  - But also within the text itself (e.g., **at the beginning of each section**, or right at the end to make a transition to the next section)

Example

4. System Design

We now detail the design of the TPS **to guarantee the Atomicity, Consistency, Isolation and Durability properties of transactions**. Each of the properties is discussed individually. We then discuss the membership mechanisms **to guarantee the ACID properties even in case of LTM failures and network partitions.**

4.1 Atomicity

The Atomicity property requires that either all operations of a transaction complete successfully, or none of them do. To ensure Atomicity, for each transaction issued, CloudTPS performs two-phase commit (2PC) across all the LTMs responsible for the data items accessed. As soon as an agreement of “COMMIT” is reached, the transaction coordinator can simultaneously return the result to the web application and complete the second phase [25].

[. . .]

Present the problem before the solution

- Always explain what the problem is **before** you present the solution!
- Hypothetical example:

Our system relies on the use of adaptive timers. When an event occurs, a sophisticated algorithm allows us to select the right value to use for the corresponding timer. This allows to avoid instability in the system. **Question from the reader:** *which instability are you talking about?*	$\Rightarrow$	The use of constant timer values may however create instability in our system: when a burst of events occur, (*something bad happens*). Our system therefore uses adaptive timers. Each time an event occurs, a sophisticated algorithm allows us to select the right value to use for the corresponding timer.

Table of Contents

Avoid passive tenses

- Passive tense is perfectly fine in many languages (French, Polish, etc.) but **it feels really bad in English!**

"For the purpose of this thesis, it was essential to build a grid simulator [...]. The Distributed Grid Simulator was built in Java [...]. Several simulations took place and let to interesting conclusions."	$\Rightarrow$	*"For the purpose of this thesis I built a Distributed Grid Simulator in Java. It allowed me to carry a wide range of simulations that lead to interesting conclusions."*

  - **Who** built the simulator and ran simulations?

Do not beat around the bush

- Write **short, straightforward sentences**

"In this section we will make a survey of the existing techniques and try to give some evaluation of their characteristics."	$\Rightarrow$	*"This section discusses existing techniques."*

Write self-contained sentences

"Modern application require increasing quantities of resources. **Especially enterprise applications, which often serve hundreds of users simultaneously.**"	⇒	"Modern application require increasing quantities of resources. This is particularly true for enterprise applications, which often serve undreds of users simultaneously."
"Detecting machine failures can be realized by the applications themselves if they use our monitoring library. **Or by letting applications select an external availability monitor.**"	⇒	"Detecting machine failures can be realized by the applications themselves if they use our monitoring library. Another option is to let applications select an external availability monitor."

Native language issues: French speakers

- This parameter **depends of** the way. . . ⇒ **depends on**
- The message may **eventually** be lost. . . ⇒ **possibly**
  - "eventually" ≠ "éventuellement"!
- Wikis are **actually** the most popular form of collaborative editors. . . ⇒ **currently**
  - "actually" ≠ "actuellement"!

Native language issues: Dutch speakers

- **When** I'm correct, you are checking the new CS-webtext ⇒ **If** I am correct. . .
  - **Probability** ≈ 1 ⇒ **"when"**: "when the train stops, passengers can get out"
  - **Probability** $\ll 1$ ⇒ **"if"**: "if the train crashes, passengers should exit through the windows"
- The local peer set is refreshed only when peers disconnect due to client exit, connection failure or long period of inactivity.
  Thus making the initial peer selection particularly critical to provide the peer with a good set of nodes to choose from.⇒ **This makes** the initial peer set. . .

Native language issues: Indians

- A decision on **the same** is yet awaited ⇒ A decision on **this matter** is yet awaited

Native language issues: Polish speakers

- The main goal of BitTorrent**,** is to distribute content ⇒ The main goal is...
- Our system is **basing** on the observation that... ⇒ **based on**
- When peer connects to central server, messages are sent back to other connected peers ⇒ When **a** peer connects to **the** central server, messages are sent back to **the** other connected peers

	Specific	Non-specific
Singular	**the**	**a**
Plural	**the**	(none)

Native language issues: Romanian speakers

- Eventually ≠ Possibly
- Effectively ≠ Actually ("Effectively" means "efficiently")
- "**In what concerns** performance, we can say..." ⇒ "Concerning performance, ..."

References

1. *How to Have your Abstract Rejected*, by Mary-Claire van Leunen and Richard Lipton.
`http://www.sigsoft.org/conferences/vanLeunenLipton.htm`
2. *The Science of Scientific Writing*, by George D. Gopen and Judith A. Swan.
`http://www.amstat.org/publications/jcgs/sci.pdf`
3. *The Elements of Style*, by William Strunk Jr.
`http://www.bartleby.com/141/`