Formatting Submissions for a USENIX Conference: An (Incomplete) Example

Your N. Here
Your Institution

Second Name
Second Institution

Abstract

Your abstract text goes here. Just a few facts. Whet our appetites. Not more than 200 words, if possible, and preferably closer to 150.

1 Introduction

There is an increasing tension between the desire to share data online, and the security concerns it entails. This paper asks the question: how can we enable *mutually distrustful parties* to consistently and reliably share data, while minimizing centralization?

The ability to share data online offers exciting opportunities. In banking, systems like SWIFT enable financial institutions to quickly and accurately receive information such as money transfer instructions; and in manufacturing, online data sharing can improve accountability and auditing amongst the globally distributed supply chain.

Increased data sharing, however, raises questions of how to *decentralize trust*. Banking institutions must currently place their trust in the centralized SWIFT's network to issue payment orders. Sometimes there is even no identifiable source of trust. Consider the supply chain for the latest iPhone: it spans three continents, and hundreds of different contractors []; neither Apple nor these contractors trust each other, yet all must be willing to agree and share information about the construction of the same product.

Recognizing this challenge by both the research and industry communities, much effort has focused on enabling shared computation between mutually distrustful parties, in the context of byzantine fault tolerance (BFT), and blockchains. Systems proposed in the literature of BFT[][] provide the abstraction of a totally ordered log; the log is agreed upon by the n participants in the system, of which at most f can misbehave. Each participant executes operations that may touch one to multiple objects in the log. In the blockchain world, Bitcoin and Ethureum have become popular distributed computing platforms providing the same log abstraction and aiming for decentralizing trust. Furthermore, Microsoft Azure has launched projects[] that leverage these blockchain platforms and extend the digital transformation beyond the companies' four walls in a supply chain.

This paper argues that there exists a fundamental mismatch between the implementation of a totally ordered log and the reality of much large-scale distributed processing. Many large-scale distributed systems consist primarily of unordered and unrelated operations. For example, a product supply chain consists of many concurrent steps that do not require ordering. Imposing an ordering on non-conflicting operations is not only often unnecessary, but costly: participants in the shared computation must vote to order operations, store the full state of the system, and replay the full log for auditing.

While there exists work on mitigating this scalability bottleneck through sharding [], the latent total order requirement introduces unnecessary coordination overhead, as coordination is performed twice, at the level of individual shards, and across shards. Callinicos [] and Omniledger [], for instance, runs a full BFT protocol for every operation. This is especially problematic when workloads are geo-replicated [], or when, as in BFT, the replication factor is high. Further, these systems support transactions under the assumption that their read and write operations are known a priori, which limits the set of applications that they can support.

As another research trend of mitigating the scalability bottleneck, EPaxos[], TAPIR[] and CURP[] only consider the ordering between potentially conflicting operations, instead of commutative operations. However, these systems assume the crash-failure model and are non-trivial to be extended to the Byzantine model, so that they cannot directly solve the problem of data sharing among mutually distrustful parties.

Existing research, in essence, is either attempting to build concurrency control and sharding functionalities over BFT replication, or integrating these functionalities into a crash-failure replication protocol. In this paper, we will show how to build these desiring functionality inside a BFT replication protocol. Specifically, our goal is to *provide the illusion of a centralized shared log, rather than the non-scalable reality of*

a totally ordered log.

2 Footnotes, Verbatim, and Citations

Footnotes should be places after punctuation characters, without any spaces between said characters and footnotes, like so.[1] And some embedded literal code may look as follows.

```
int main(int argc, char *argv[])
{
    return 0;
}
```

Now we're going to cite somebody. Watch for the cite tag. Here it comes. Arpachi-Dusseau and Arpachi-Dusseau coauthored an excellent OS book, which is also really funny [**?**], and Waldspurger got into the SIGOPS hall-of-fame due to his seminal paper about resource management in the ESX hypervisor [**?**].

The tilde character (~) in the tex source means a non-breaking space. This way, your reference will always be attached to the word that preceded it, instead of going to the next line.

And the 'cite' package sorts your citations by their numerical order of the corresponding references at the end of the paper, ridding you from the need to notice that, e.g, "Waldspurger" appears after "Arpachi-Dusseau" when sorting references alphabetically [**?**, **?**].

It'd be nice and thoughtful of you to include a suitable link in each and every bibtex entry that you use in your submission, to allow reviewers (and other readers) to easily get to the cited work, as is done in all entries found in the References section of this document.

Now we're going take a look at Section 4, but not before observing that refs to sections and citations and such are colored and clickable in the PDF because of the packages we've included.

3 Floating Figures and Lists

Here's a typical reference to a floating figure: Figure 1. Floats should usually be placed where latex wants then. Figure1 is centered, and has a caption that instructs you to make sure that the size of the text within the figures that you use is as big as (or bigger than) the size of the text in the caption of the figures. Please do. Really.

In our case, we've explicitly drawn the figure inlined in latex, to allow this tex file to cleanly compile. But usually, your figures will reside in some file.pdf, and you'd include them in your document with, say, \includegraphics.

[1] Remember that USENIX format stopped using endnotes and is now using regular footnotes.

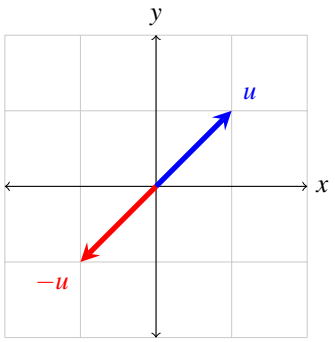

Figure 1: Text size inside figure should be as big as caption's text. Text size inside figure should be as big as caption's text. Text size inside figure should be as big as caption's text. Text size inside figure should be as big as caption's text. Text size inside figure should be as big as caption's text.

Lists are sometimes quite handy. If you want to itemize things, feel free:

fread a function that reads from a `stream` into the array `ptr` at most `nobj` objects of size `size`, returning returns the number of objects read.

Fred a person's name, e.g., there once was a dude named Fred who separated usenix.sty from this file to allow for easy inclusion.

The noindent at the start of this paragraph in its tex version makes it clear that it's a continuation of the preceding paragraph, as opposed to a new paragraph in its own right.

3.1 LaTeX-ing Your TeX File

People often use `pdflatex` these days for creating pdf-s from tex files via the shell. And `bibtex`, of course. Works for us.

Acknowledgments

The USENIX latex style is old and very tired, which is why there's no \acks command for you to use when acknowledging. Sorry.

Availability

USENIX program committees give extra points to submissions that are backed by artifacts that are publicly available. If you made your code or data available, it's worth mentioning this fact in a dedicated section.