TnT : A file system synchronizer [†]

Justin Holmgren holmgren@mit.edu
Zach Kabelac zek@mit.edu
Pritish Kamath pritish@mit.edu
Deepak Vasisht deepakv@mit.edu

May 11, 2014

Abstract

We build a peer-to-peer file-system synchronizer, based on Tra, which is performant and resilient to system crashes. Slightly more stuff needed.

1 Introduction

2 Goals

We plan to implement a file synchronization system based on Tra. We plan to ensure that we achieve all the five objectives mentioned in the introduction of Tra Technical Report [1]. We will implement multiple versions starting from a basic version that just ensures correctness (including Tra's "no false conflicts" guarantee) and optimize further for reduced network communication, and reduced metadata overhead. We plan to test it on up to 10 machines (or virtual machines, maybe on EC2) running concurrent updates to file systems.

3 Implementation Details

We will use Go for our implementation because of its good support for OS-level operations like watching the file system, high-level programming features, as well as our experience using Go in 6.824.

We will iteratively improve our file synchronization system, starting with a minimum viable implementation. In particular, for our first iteration we will only try to achieve correctness using essentially the same techniques (the Vector Time pair algorithm) as in the Tra, but we will not implement any nonessential optimizations.

If we have more time, we will add the optimizations described in [1].

[†] **TnT** stands for "**T**nT is **n**ot **T**ra"

4 Project Evaluation

We will test the system developed on up to 10 concurrent machines for:

- ◇ **Correctness**: We will test the case where machines modify disjoint sets of files, while reading all the files, and we will repeat this while changing the sets of files written by each machine, so that each file is at some point written by every machine. There should be no conflicts in this scenario, so we will check this, as well as checking that each machine has the same copy of each file. We will also check that whenever two machines concurrently update the same file in different ways, a conflict is reported.
- ◇ **Metadata overhead**: the metadata used at any point of time should be proportional to the size of the file system tree. if files are deleted by all the synchronizing peers, they should not contribute to metadata.
- ◇ **Communications overhead**: the communication used should be proportional to the number of files changed, and the sizes of the files. In particular, it should not be proportional to the size of the file system tree.

References

[1] Russ Cox, William Josephson. Tra Technical report