MIT OpenCourseWare
http://ocw.mit.edu

6.033 Computer System Engineering

Spring 2009

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.

Preparation for Recitation 16

Read *How to Build a File Synchronizer* by Trevor Jim, Benjamin Pierce and Jerome Vouillon. (This unpublished paper is not in the handouts.) You can find out more information about Unison at the The Unison Home Page. A User Manual and Reference Guide is also available. (You may wish to read Section 10.4 of the textbook before reading the Unison paper --- Section 10.4 shows the essence of the ideas, while the Unison paper shows many of the real-world problems that crop up when attempting to implement them!)

Unison is designed to solve the problem of reconciling two file repositories when both are subject to asynchronous changes. The industry uses two terms to describe this process --- "synchronization" and "reconciliation." In 6.033 we will try to use the term "reconciliation" for this process and reserve the term "synchronization" for processes that involve setting two or more clocks to the same time.

A common use of Unison is to maintain consistency between a set of files in your Athena home directory and a laptop. Each time you run Unison, the program would:

- Discover new files added to the Athena directory and copy them to the laptop
- Discover new files added to your laptop and copy them to your Athena directory
- Discover files deleted from your Athena directory and delete them from your laptop
- Discover files delete from your laptop and delete them from your Athena directory
- Propagate any metadata changes from one system to the other (ie: changes in file modification time)
- Make an attempt to inform the user of its proposed actions and receive approval
- Update everything atomically without losing any data in the process

As the authors note, it is rather straightforward to build a simple file reconciler; it is much harder to build one that is efficient over slow links, that can work across operating systems, and that is tolerant of failures in either the systems being reconciled or the network.

Reconciling files between multiple computers is a longstanding problem among computer scientists, born from the practical problems of maintaining a consistent environment across multiple machines. Early versions of Berkeley 4.2 Unix (circa 1984) came with a one-way file reconciler called **rdist**. This program inspired Tridgell [PhD Thesis] to write **rsync**, a program that performed much the same function, but improved performance with the "rsync algorithm" --- a technique for incrementally updating large files such as logfiles that tend to be extended rather than rewritten. Both **rdist** and **rsync** are more properly thought of as efficient publication systems rather than reconciling systems: they will propagate changes from a central computer to other nodes across a network, but they are not very good at bi-directional reconciliation. Both of these programs are further limited in that they only operate under Unix, while Unison runs under Windows and MacOS as well.

Russ Cox and William Josephson created a multi-host file synchronizer called tra which is pretty neat, but it only runs on Unix.

As the authors of the paper make clear, much of their effort has been spent on making Unison work in a cross-platform manner. For example, Unix file modification times have a resolution of 1 second, while Windows file modification times have a resolution of 2 seconds. The solution that Unison takes is to mask the bottom bit of the Windows file modification times. One of the difficulties in writing Unison is that the authors needed to discover these inter-operating system differences by trial and error: there is no single document that lists them all.

Unison is unquestionably a technical success: the program is included in multiple Unix distributions and members of the 6.033 staff have been using it since 1998. Nevertheless, Unison does have its drawbacks, some of which can be inferred from the paper:

- In Section 2, Jim, Pierce and Vouillon argue that "the most important goal of any file synchronizer is safety," but then they point out in Section 4 that if Unison cannot read the files in one file archive due to a hardware failure, it will proceed to delete all of the files in the other archive. Obviously, this is precisely the wrong thing to do (despite the authors' claim that Unison had only experienced this catastrophic failure once, rest assured that it happens with some regularity). Why do you think such catastrophic failures happen, and how could Unison be modified to make them less likely? What 6.033 design principles does Unison violate which makes such catastrophic failures more likely?
- Unison is a pairwise reconciler; what if you have three computers that you want to keep in sync? One way to set up such a network is to have A reconcile with B and B reconcile with C. This actually works in practice (a point that the authors fail to make). What do you think happens if you have C in turn reconcile with A? Will updates be propagated more quickly, or will the system fail to converge to a stable state?
- Unison generally fails when a file in one repository cannot be adequately represented in another one. For example, a file that has a colon in its filename on Unix cannot be reconciled onto either a Mac or a Windows file system. Likewise, the author note that "there is no system call to find out the maximum length of file names in a (possibly remotely-mounted) directory." (p. 8). Do you believe that these limitations are inherent in the construction of a multi-platform file reconciler, or do you think that they could have been detected and coded around?

The authors of the paper argue that they have taken several steps to improve performance, but in their table at the top of page 9 (Figure 2) they note that their program is slower in every case than Rsync. Why do you think this is so? What else is wrong with the information provided in the table? Despite being compared with rsync, Unison cannot interoperate with it. Do you think that this was a good design decision?

After reading the paper, would you trust Unison to reconcile your files? Why or why not?