Nathan Matsuda - CS443 - 2/21/07

1. Paper title and its author(s).

Replay Debugging for Distributed Applications
Dennis Geels Gautam Altekar Scott Shenker Ion Stoica

2. Brief one-line summary.

The paper introduces liblog, a tool for debugging distributed applications by logging program execution for later offline replay.

3. A paragraph of the most important ideas: perhaps a combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.

Distributed applications can be debugged for invalid memory access and whatnot using traditional methods, but a considerable aspect of their execution is determined by the actual distributed environment and the interaction between many isolated nodes. The authors propose a system which logs all libc calls, which include file access and ingoing/outgoing network messages. These logs can then be loaded into a debugger, which can be remotely located, and replayed to isolate the bug.
The authors note that there are some severe limitations to the system. First, it only works for calls that happen to libc. Context switches between threads can alter resource states without liblog's knowledge. To work around this, liblog blocks context switches until libc calls, so that it will always have a consistent view of system resources. The logging itself uses a large amount of disk space, since the logs contain all incoming and outgoing data for the application. There is also a modest network overhead because messages are timestamped for the debugger to replay network traffic. Bugs which cause data to be written into memory used by liblog could cause unknown behavior that could ruin any logged data.
The authors state that they intend this system to be usable as an always-on layer under actual distributed applications. They note that if a debugger is run with a system and then removed, the removal itself can potentially cause bugs that would not be possible while the debugger was still running. To illustrate the cost of running liblog constantly, the authors present a few graphs showing bandwidth overhead and log growth rate, etc. It is difficult to evaluate this impact because, although there is clearly a strong impact from the use of liblog, there are no alternatives to this type of system.

4. A paragraph of the largest flaws; maybe an experiment was poorly designed or the main idea had a narrow scope or applicability. Being able to assess weaknesses as well as strengths is an important skill for this course and beyond.

This type of distributed logging for deployed applications seems very useful, but this implementation of it doesn't seem to meet the author's intent of being useable in real world situations. The system is limited to applications that don't rely heavily on multithreading or heavy data streaming. The authors also don't go into great detail on log access in a wide distribution would be handled, nor how a debugger would deal with the vast number of concurring execution states in a widely distributed application. It seems beyond human capability to discern a bug that is the result of more than a few concurrent events on as many separate machines.

5. A last paragraph where you state the relevance of the ideas today, potential future research suggested by the article, etc.

As I just mentioned, there is clearly a use for this type of debugging, given the growth of production p2p systems in deployment, like bitTorrent, Skype, and Joost. The authors make a good point that there is a broad class of bugs for these systems that cannot be resolved using traditional methods.