[Title]

[The Problem]
- Try to automatically figure out which set of files are needed to deploy a software service
- Programmer develops a service on a developer box
- Experiments with different middleware
- Makes ad-hoc changes to configuration
- Gets software running

[the problem—difficulties]
- Must then replicate this configuration onto a server machine in a remote datacenter
- Easiest—replicate entire machine on server (security risk, license problems, etc)
- Want to deploy a minimum set—call this a deployment set--but what is this minimum set?
- Goal: a little tool that can automatically determine what the deployment set is

[Inspiration]
- Look at software configuration management and source code management and see if there are any interesting crossovers
- In particular, Vesta system, an advanced source code management system
- Exports file system via NFS to track build dependencies automatically

[Different approaches]
- How do we gather information needed to calculate a deployment set?
- Runtime: Can see exactly which files used, but some files are necessary and not used, also hard to determine meanings of files
- Binary: Go through binary, extract out parts of file paths, try to piece together names of other files. Interesting problem algorithmically but entirely impractical
- Source Code: Might be possible, but access to source code is difficult to get, so probably not workable from a practical standpoint
- Metadata: Get the most direct and accurate information. In fact, some metadata is already available in RPMs, but not really immediately practical
- Let’s use a runtime approach

[Gathering runtime data]
- Trade-offs of information detail vs. generalizability/ease of implementation
- i-node
- system call
- application specific
- Disk blocks?—doesn’t really fit in spectrum. Some interesting possibilities

[Gameplan]
- Log system calls to determine file usage of the system
- Track process and file dependencies
- User specifies processes that are needed by final system
- Tool will examine log data to determine as small a sized deployment set as possible

[Let’s get started]

- Fedora Core 3
- Configured as a web development system (3.96 GB)
- Logging system calls is annoying—do it User Mode Linux (entire Linux HD is copied to a virtual Linux drive with almost no changes)—this mostly works
- UML makes it easy to test deployment set as well
- Track process creation, exit, execve, file open, close, socket stuff

[Dependency Framework]

- Building a framework for pruning
- Must track dependencies of processes and files, then we can figure out what is needed and what’s not needed
- Process dependency on fork, exec, socket connect
- File dependency on file read, opposite dependency on write
- Add time information—not used

- Specify which process (or a process and children) to include in deployment set
- Tool will figure out what else is needed

[Framework Operations]

- Here are queries and operations that the framework allows you to do
- …

[Evaluation and Analysis details]

- Lots of subtle details that make a normal evaluation impossible
- Usual approach: Here’s my amazing algorithm, here’s a bunch of test instances, and look how well my algorithm works

[Good Results/Bad Results]

- We are interested in the overall, general performance of this approach
- However, due to weaknesses of the underlying model—monitoring of runtime behaviour and analysis of system call behaviour of an application—it is always possible to construct a particular, pathological example that will cause some sort of failure in the deployment set generated by the tool (either too big or doesn’t work)
- So failures of the tool are not really meaningful because they might not be indicative of typical behaviour
- Similarly, for any particular example, it is possible to tweak the algorithm to run well and without unusual failures on that example
- So successes of the tool are not particularly meaningful because I can take any test suite, tune the algorithm to run perfectly on that test suite, and then claim that my tool works perfectly even though the tool will not behave this well typically

[What to do?]
- Problem similar to this occurs in machine learning
- Behaviour of an algorithm on a test suite is not meaningful because it does not give any indication about generalization performance on the algorithm
- Usually, you create multiple test sets to deal with this problem—but creating test sets is difficult in this particular instance

- Instead, I did most of the coding ahead of time
- I've created two test instances, will run the tool on these two tests
- Will document the issues that arose and any tool changes made
- The documented problems will be suggestive of potential issues that might arise when the tool is used in more difficult situations
- It will give an idea of limitations and usefulness of the tool
- But the actual results themselves are not significant (as mentioned previously, it would have been fairly easy to tweak the algorithms with application-specific hacks etc. to run perfectly on these instances, but such results would be misleading. So the results should not be evaluated as being bad or good.)

[Reality]
- I'm lazy

[Two test sequences]
- Login as root
- ls
- halt

- Login as root
- ifconfig eth0 192.168.1.3
- access a Php page that uses a MySQL database (assume Postgres DB will not show up in deployment set)
- halt

[Sanity Test]
- Take all files used during a test sequence as the deployment set
- Problem: certain system calls not logged: readlink, chdir, stat64
- (In particular, certain empty directories were not included in the deployment set)

[Strict Dependencies]
- Try: strict dependencies
- (follow all dependencies from the previously specified processes, include everything visited)
- Problem: some files are read and written to by a lot of key processes, lots of things are included that shouldn't be e.g. /etc/mtab /var/run/utmp /dev/console etc.
- Manually remove those
- Certain system processes that configure system and devices are not running

[ls Dependencies picture]
[Details of problems]
- MySQL does not require "Configure system devices" but system devices need to be configured before it works
- Bash does not need ftpd or Oracle

[Dependencies with System files]
- Add descendants of /etc/rc.d/rc.sysinit, /sbin/runlevel, processes with no binary
- Mostly works
- But problems with things that you exec that aren't important but program will fail if the exec fails
- Can just include stub binaries
- Instead, will include all sub-processes, but not those of /etc/rc.d/rc (Linux start-up process—tolerant of exec failures)

[More details]
- Tee is used to output log data to multiple places, but log data is never used

[Dependencies with Exec'd files]
- Problem: certain files are written to but never read (i.e. log files), but those files cannot be created because parent directories not created—easy to fix later

[Summary Table]

[Conclusions]
- This research suffers from some fundamental problems
- Take a Niche problem that no one is really interested in
- Try the Obvious solution
- Get the Expected answers

[Conclusion]
- In other words:
- Tool not entirely automatic
- Needed user intervention to identify errors and apply simple fixes
- More useful to have a visualization tool with some automatic features that allow tweaks of stuff

[Future work]
- Multiple machines (handle deployment sets of services that span multiple machines, handle alterations of deployment set files to refer to different machines, etc)
- Multiple platforms (Windows, Java, MacOS)
- Deployment set diffing and merging (e.g. deployment set for a Linux upgrade merged with a deployment set for a program or trying to figure out what changes you have made from the default configuration)
- Visualization stuff (In a non-automatic tool, visualization becomes important—e.g. profilers etc)

- Research-wise, Meta-data stuff might be interesting—what's the ideal meta-data for doing stuff like this and how can it be auto-generated (but again, it's completely impractical)