The Maelstrom: Network Service Troubleshooting Via “Ineffective Procedures”

Alva L. Couch, couch@eecs.tufts.edu
Noah Daniels, ndaniels@eecs.tufts.edu
Tufts University, Medford MA USA
http://www.eecs.tufts.edu/~couch/maelstrom
Target problem: automate network troubleshooting.

Starting point: list of services to assure.

Easy part: how to assure one service.

Hard part: precedences between assurance tasks.
Dreams

- Quicker response to network problems.
- More collaboration.
- Less “boring” work.
- Acceptable losses!
Silly Example

A: network up → E: nfs server up

B: nis bound

D: home dirs available

C: commands work
Ideal: “Plug and Play” Automation

- Grab the scripts that you need from others.
- Scripts all just “get along” and work together.
- Make script writers work harder.
- So administrators’ work is easier!
Let’s Automate!

- Suppose we write scripts A, B, C, D, E to check and repair corresponding functions.
- Normally, we’d have to remember to run them in the order “A B E D C”.
- We’d usually do that by predeclaring precedences: B:A means “B must follow A”.
Predeclaring Precedences

A: network up  E: nfs server up
B: A  D: A  E: A  D: E  E: A
B: A  D: A  E: A
C: B  D: E
B: nis bound  D: home dirs available
C: B  D: E
C: commands work
B: A  C: B  C: D  D: A  D: E  E: A
Predeclaring Precedences Is a Pain!

- Must **know** precedences beforehand.
- Must **update** precedences whenever you add or remove a script.
- In some cases, precedences are **unknown** or **dynamic**!
- In this case, **any fixed order is an ineffective procedure** for troubleshooting some problems.
One Ineffective Procedure

F: filesystem OK

G: fsck command in filesystem OK

No fixed order will ensure success.

“Chicken and Egg” problem!
Discovering Order

- Suppose that A, B, C, D, E are crafted so that:
  - They fail robustly when called at the wrong time.
  - They tell you when they fail.
  - They don’t undo each other’s actions.
- Then we may infer their required execution order from their behavior rather than declaring precedences beforehand!
Permutation Embedding

- For a set of objects $x_1, x_2, \ldots, x_n$, all permutations of the objects are embedded in the string containing $n-1$ copies of $x_1, \ldots, x_n$, followed by $x_1$.

- E.g., $x_1, \ldots, x_n, x_1, \ldots, x_n, \ldots, x_1, \ldots x_n, x_1$

  n-1 copies trailing $x_1$
Example of Permutation Embedding

<table>
<thead>
<tr>
<th>Embedding</th>
<th>Permutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDEABCDEABCDEABCDEA</td>
<td>ABCDE</td>
</tr>
<tr>
<td>ABCDEABCDEABCDEABCDEA</td>
<td>BACDE</td>
</tr>
<tr>
<td>ABCDEABCDEABCDEABCDEA</td>
<td>ACBDE</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ABCDEABCDEABCDEABCDEA</td>
<td>ECDBA</td>
</tr>
<tr>
<td>ABCDEABCDEABCDEABCDEA</td>
<td>DECBA</td>
</tr>
<tr>
<td>ABCDEABCDEABCDEABCDEA</td>
<td>EDCBA</td>
</tr>
</tbody>
</table>
Exploiting Permutation Embedding

- Don’t record precedences.
- Try scripts in embedding sequence.
- Record successes.
- Don’t repeat trials that succeed.
- Retry scripts that fail.
- Until all succeed!
Discovering Order (1)

Precedences

A → E
B → D
C

Execution order

A
B C D E
A B C D E
A B C D E
A B C D E
A

Discovered order:
A
Discovering Order (2)

Precedences

A → E

B → D

C

Execution order

A B C D E

A B C D E

A B C D E

A B C D E

A B C D E

A

Discovered order:

A B
Discovering Order (3)

Precedences

A → E
B → D
C

Execution order

A B C D E
A B C D E
A B C D E
A B C D E
A

Discovered order:

A B
Discovering Order (4)

Precedences

A → E
B → D
C

Execution order

A B C D E
A B C D E
A B C D E
A B C D E
A

Discovered order:
A B
Discovering Order (5)

Precedences

A → E
B → D
C

Execution order

A B C D E
A B C D E
A B C D E
A B C D E
A

Discovered order:
A B E
Discovering Order (6)

Precedences

A → E
B → D
C → B

Execution order

A → B → C → D → E
A → B → C → D → E
A → B → C → D → E
A → B → C → D → E

Discovered order:

A B E
Discovering Order (7)

Precedences

Execution order

Discovered order: A B E
Discovering Order (8)

Precedences

Execution order

Discovered order: A B E
Discovering Order (9)

Precedences:

A → E
B → D
C

Execution order:

A B C D E
A B C D E
A B C D E
A B C D E
A

Discovered order:

A B E D
Discovering Order (10)

Precedences:

A → E
B → D
C → C

Execution order:

A B C D E
A B C D E
A B C D E
A B C D E
A

Discovered order:

A B E D
Discovering Order (11)

Precedences: A → E, B → D, C → B → D

Execution order:
- A
- B
- C
- D
- E

Discovered order: A B E D
Discovering Order (12)

Precedences

Execution order

Discovered order:

A B E D
Discovering Order (13)

Precedences

Execution order

Discovered order:

A B E D C
Effect of Initial Ordering

- Efficiency depends upon initial ordering of tasks.
- **Best case**: initial order is appropriate order.
- **Worst case**: initial order is opposite to appropriate order.
Best Case: ABEDC

Precedences

A → E

B → D

C

Execution order

A B E D C

A B E D C

A B E D C

A B E D C

A B E D C

A B E D C

Discovered order:

A B E D C
Worst Case: CBDEA (1)

Precedences

A \rightarrow E
B \rightarrow D
C

Execution order

C B D E A

Discovered order:

A
Worst Case: CBDEA (2)

Precedences

Execution order

Discovered order:

A B E
Worst Case: CBDEA (3)

Precedences:

- A → E
- B → D
- C → B, D

Execution order:

- C B D E A
- C B D E A
- C B D E A
- C B D E A

Discovered order:

- A B E D
Worst Case: CBDEA (4)

Precedences

A → E

B → D

C

Execution order

C B D E A
C B D E A
C B D E A
C B D E A
C B D E A

Discovered order:

A B E D C
How Bad Can Bad Get?

Precedences

Execution order

Discovered order:

A B C D E
Small Errors Mean Small Inefficiencies

Precedences

A → E
B → D
C

Execution order

A B D E C
A B D E C
A B D E C
A

Discovered order:

A B E D C
Implementation: The Maelstrom

- **mael** command is a dispatcher for a set of scripts.
- Input is a list of commands to try.
- **Mael** tries to make them all **succeed** (exit code 0).
- Nonzero exit code means **failure**; try again.
Seeding the Storm

- Can give **mael** hints and other information about its commands.
- **B:A** - I think command **B** should be tried after **A**.
- **B::A** – **B** cleans up after **A**. **B** must be retried if it succeeds before **A**.
- **B:::A** – I know **B** will only succeed after **A**.
Command Requirements

- Maelstrom only functions correctly if the commands that it dispatches are:
  - **Aware**: commands know whether they failed.
  - **Homogeneous**: commands that change the same system attribute change it in the same way.
  - **Convergent**: commands that discover that goals are already met do nothing.)
How Difficult Is It to Write a Conforming Maelstrom Script?

- Easy part: **awareness**.
  - **Local** to the script.
  - Insert enough branches to check for script preconditions.

- Hard part: **homogeneity**.
  - **Global** convergence criterion.
  - All scripts must agree on desired effects.
Form of Maelstrom Script

- **Check all preconditions** necessary for script function.
- If preconditions are not present, **fail**.
- Else try to **fix a problem**.
- If that seems to work, **succeed**.
- Else **fail**.
Engineering Maelstrom Scripts

- No preconditions for the script as a software unit.
- Safe to run in any sequence with other scripts.
- Only thing in doubt: homogeneity.
- Do scripts agree on what to do?
Imperfect Storms

- ::, ::: help compensate for imperfect command behavior.
- A::B – A and B aren’t homogeneous and A should be done last, even if B succeeded last.
- A:::B – A isn’t aware that it needs B, so do B first.
A Lesson Learned

- **Causality** is a **myth** in a sufficiently complex system.
- Cannot determine what will happen in general.
- Can determine what repaired a specific problem.
- This is not the same as what caused the problem.
Not Causal, but Operational

- Impossible to determine true precedences between tasks by direct observation (Sandnes).
- Easy to determine an order that satisfies unknown precedences.
But Wait, There’s More!
In the Paper:

- **Decision trees** represent best practices.
- **Mael**’s commands can represent decision trees.
- **Mael** replaces **make**’s global precedence knowledge with dynamic probes during commands.
- Can implement **make in mael**.
Status and Availability

- [http://www.eecs.tufts.edu/~couch/maelstrom](http://www.eecs.tufts.edu/~couch/maelstrom)
- Platform: Perl 5.
- Portable to most any system.
- Intensively tested on a “precedence simulator” that simulates behavior of troubleshooting scripts.
- Working on script content now.