# Architecting Failovers

Peter Juritz
peter@nimbula.com

nimbula

- IaaS software for private clouds
- APIs for provisioning and managing infrastructure at scale

## What is a failover?

According to wikipedia:

“In computing, failover is automatic switching to a redundant or standby computer  server, system, hardware component or network upon the failure or abnormal termination of the previously active application, server, system, hardware component, or network.”

## Put more simply

If something breaks, try to fix it automatically.

## Who needs failovers?

- Failures WILL happen
- Highly available services
- Operation critical services
- Environments which expect failure
    - At larger scale, expect more failures
- Remember, this is not your main goal

## Engineering failovers can lead to Unforseen Consequences

## What can cause failures?

- Very difficult to predict, but:
- Bugs in code, both
  - Code you have written
  - Code you rely on

## What can cause failures?

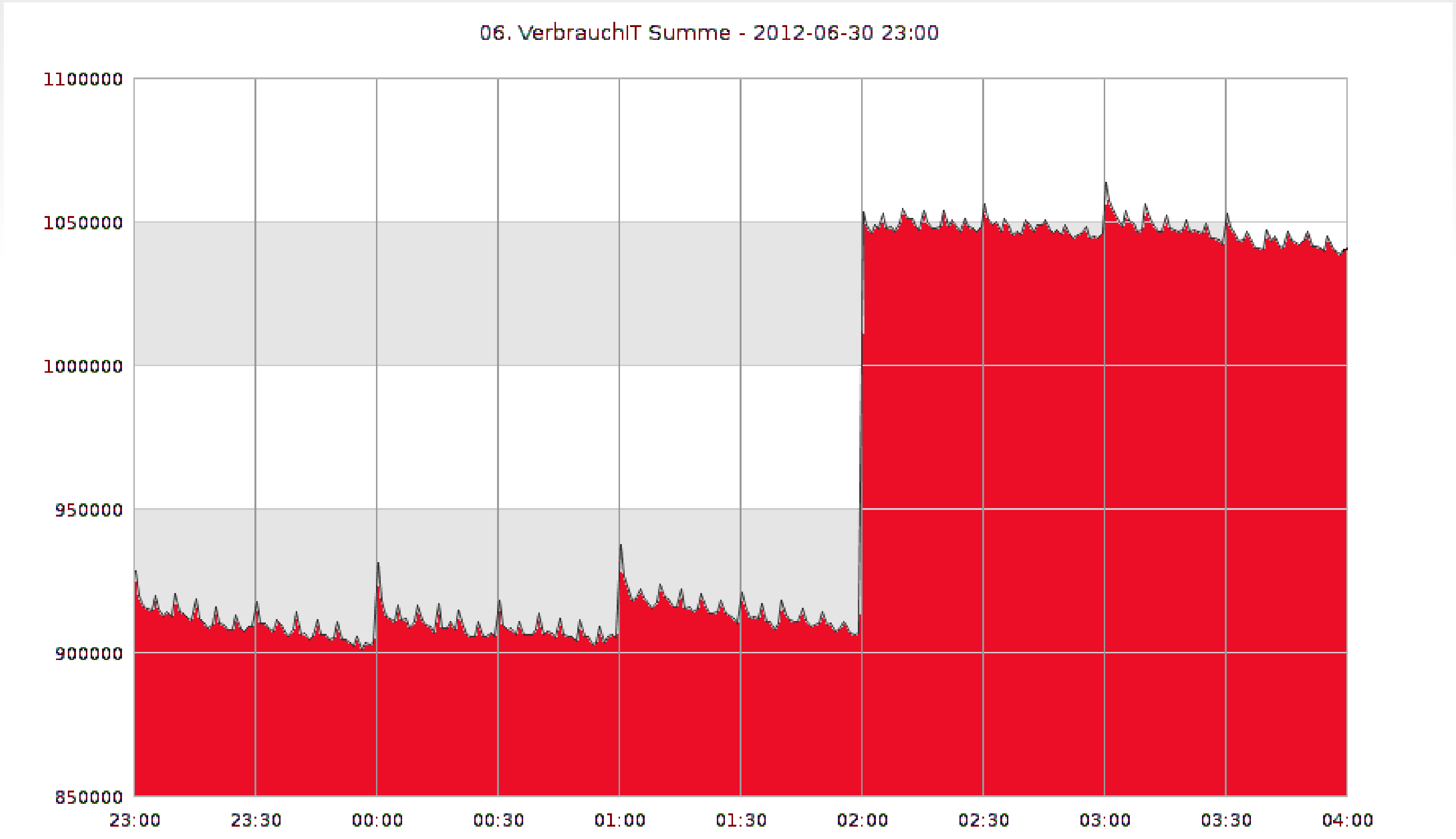
- Hardware failures
  - Network
  - Disk
  - Power Outages

## What can cause failures?

- The Human Element

# Some examples from experience

- The linux leap second bug



## Some examples from experience

- Users like to reboot everything

## Different cases of failure

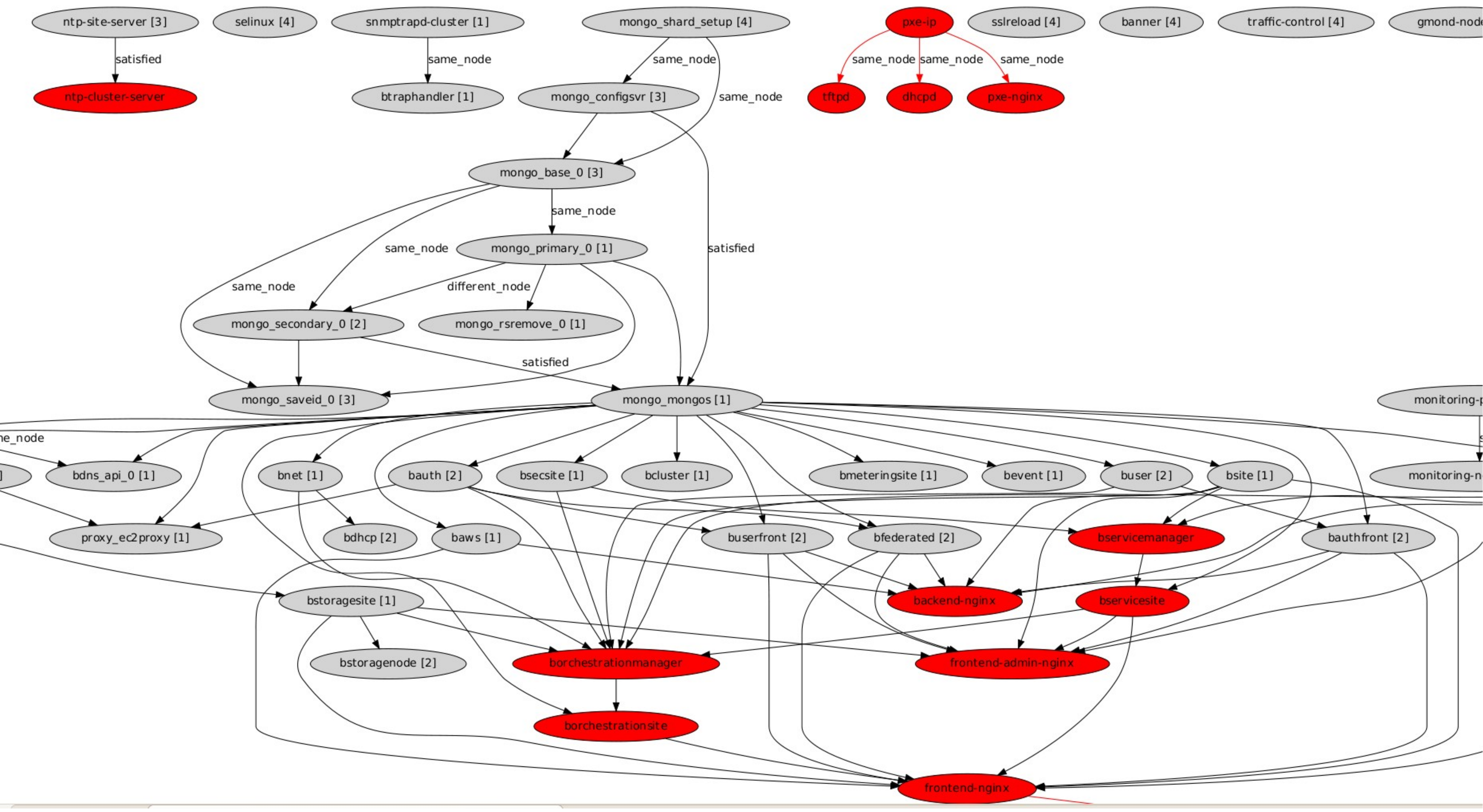
- Hardware dies
- Processes die
- Processes spin/ leak memory
- Data problems (be vary wary)
- Unrecoveravle failures

## Detecting a fault

- Not always easy
    - Watch PIDS?
    - Heartbeats? Pings? Idle connections?
    - Health checks?
    - Distributed locks?
    - Very good exception handling/ logging
- This may be the hardest part

## Some things we've done

# bIC (Infrastructure Controller)



## DNS failover

database.internal. → 10.0.0.23

**BANG**

database.internal. → 10.0.0.42

## Apache Zookeeper

- Distributed Coordination system

## Simulate Failures

- This can be hard
- You won't catch all the cases

But try! Better now than in production

## Replicated Databases

- Mongo, Riak, Cassandra...
- We happen to use

mongoDB

## Some lessons we've learnt

## Some things you can not recover from – don't waste time trying to

Simpler service architecture

=

Easier recovery

Thinking about failure forces you to examine architectural weakness

Focus your effort on the most likely problem areas