

Sloppy Management for Structured P2P services

Paolo Costa

VU University Amsterdam

Guillaume Pierre

VU University Amsterdam

Alexander Reinefeld

ZIB Berlin

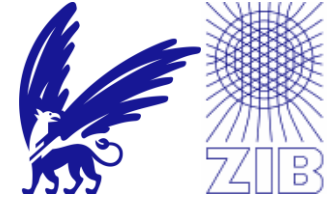
Thorsten Schütt

ZIB Berlin

Maarten van Steen

VU University Amsterdam

Programming P2P overlays is much harder than it looks



□ Take **Chord** for example

- The routing algorithm itself couldn't be any simpler
- 99% of complexity comes from overlay maintenance (build and maintain finger tables and successor lists, deal with failures, etc.)

□ Complexity means:

- Additional work (**ad-hoc algorithms**)
- Also, complex **liveness bugs** [NSDI 2007]

**Can't we make management
a bit more systematic?**

Position: Repair Algorithms Are Evil

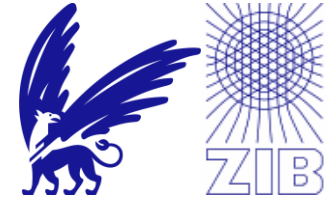
□ Repair algorithm:

```
while (true) {  
    detect_problem();  
    do_something_about_it();  
    /* Be careful about concurrent repairs,  
       inaccurate problem detection, etc. */  
}
```

□ Sloppy Algorithm:

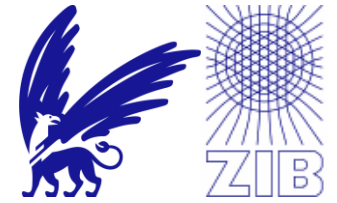
```
specify_desired_invariant();  
while (true) {  
    do_nothing_about_problems();  
    /* Trust that problems will be taken care of in the  
       background */  
}
```

Example: build and maintain Chord finger tables



- ❑ Let's redesign Chord's finger table maintenance using a 2-layer **epidemic algorithm**
 - One layer maintains **random links** between nodes (good for connectivity)
 - One layer **selects links** based on a criterion (here: which finger are you looking for?)
 - Epidemic protocols run periodically **regardless of any event in the overlay**
- ❑ **Good properties:**
 - Very fast **convergence**
 - No need to do anything about (partial) failures
 - Constant background management traffic
- ❑ **Do not worry about the algorithm's sloppiness!**
 - There will be temporary overlay inconsistencies
 - But no more than using repair algorithms

- Let's explore if we can generalize the approach to other forms of overlay management
 - Load balancing **before load problems occur**
 - Enforce **uniform distribution of overlay in-degree**
 - Try to defeat Eclipse attacks...
 - Long-term **management of Grid jobs**
 - Continuously try to identify performance bottlenecks (machine selection, I/O, network capacity, etc.)
 - Migrate processes to “better” nodes when possible
 - The optimization process should be transparent to the job! Each job is different, no need to change anything



Questions?

Bits and pieces...

- We do not have a global view of a solution yet, just bits and pieces
 - **Process migration:** rely on virtualization
 - **Location transparency:** Mobile IPv6? IP-over-P2P?
 - **Identify execution bottlenecks:** that's harder but we hope we can use gossiping
 - **Select suitable replacement nodes:** use traditional resource scheduling services