Overcoming Software Fragility with Interacting Feedback Loops and Reversible Phase Transitions

Sept. 23, 2008
BCS 08 – Visions of Computer Science

Peter Van Roy

Université catholique de Louvain
Louvain-la-Neuve, Belgium
Overview

- Motivation for interacting feedback loops
  - Example from Norbert Wiener
  - Human respiratory system
  - Software example: TCP

- Structured overlay networks (SELFMAN project)
  - We are using overlay networks for distributed applications
  - Relaxed ring: handles imperfect failure detection
  - Merge algorithm: handles network partitioning

- Physical analogy
  - Our practical structured overlay network shows phases
  - Robust software should have reversible phase transitions
Interacting feedback loops
Feedback loops

- A feedback loop consists of three elements that interact continuously with a subsystem: a monitoring agent, a correcting agent, and an actuating agent.
  - The elements and the subsystem are concurrent components interacting through asynchronous message passing.
  - The correcting agent has an abstract model of the system and a goal.
  - The model does not have to be complete but it has to be correct.

- Example: transaction manager using concurrency control.
  - monitor = resource request, actuator = resource grant/refusal, corrector = model of who has exclusive access to what resources.
Example from Wiener (1948)

- A system with two loops interacting through a common subsystem

  - This is unstable!
  - Wiener leaves the fix as homework for the reader
  - One possible solution: outer loop (tribesman) controls the other by simply adjusting the thermostat
  - One loop controls the other
Human respiratory system

- Trigger unconsciousness when O2 falls to threshold
  - Render unconscious (and reduce CO2 threshold to base level)
  - Increase or decrease breathing rate and change CO2 threshold (maximum is breath-hold breakpoint)
- Trigger breathing reflex when CO2 increases to threshold
- Trigger laryngospasm temporarily when sufficient obstruction in airways

- Breathing reflex
- Laryngospasm (seal air tube)
- Detect obstruction in airways
- Measure CO2 in blood
- Monitor breathing
- Measure O2 in blood

Actuating agents
Monitoring agents

Breathing apparatus in human body
Discussion of respiratory system

- **Four interacting feedback loops**: two inner loops (breathing reflex and laryngospasm), a loop controlling the breathing reflex (conscious control), and an outer loop controlling the conscious control (falling unconscious)
  - This design is derived from a precise textual medical description (if you believe Wikipedia: entry “Drowning” from 2006)

- **Holding your breath can have two effects**
  - Breath-hold threshold is reached first and breathing reflex happens
  - O₂ threshold is reached first and you fall unconscious, which reestabishes the normal breathing reflex

- **Some plausible design rules inferred from this system**
  - Common design pattern: one loop controlling another
  - Conscious control is sandwiched in between two simpler loops: the breathing reflex provides abstraction (consciousness does not have to understand details of breathing) and falling unconscious provides protection against instability
Software example: TCP

- This example shows a reliable byte stream protocol with congestion control (a variant of TCP)
  - This diagram is for the sending side
- The congestion control loop manages the reliable transfer loop
  - By changing the sliding window’s buffer size
- Again, an essential pattern is one loop controlling another
Structured overlay networks ("peer-to-peer")
Robust distributed systems with structured overlays

- How can one build robust distributed systems?
  - One approach is to make them decentralized and self-managing
    - No single point of failure, every node can play any role
  - A good example is the structured overlay network, which is an example of a peer-to-peer network with strong self-organizing properties
  - In the SELFMAN project we have built a practical structured overlay network, a transactional storage service on top, and a Distributed Wiki application using this service (*)

- For SELFMAN it is important to make overlay networks practical
  - Coping with imperfect failure detection and network partitioning
  - For imperfect failure detection: the relaxed ring [Mejias et al 2008]
  - For network partitioning: the merge algorithm [Shafaat et al 2008]

- We then made an observation that led to this paper:
  - Both of these contributions lead to the same physical analogy

(*) First prize in IEEE International Scalable Computing Challenge (SCALE 2008)
Structured overlay networks: inspired by peer-to-peer

- Hybrid (client/server)
  - Napster

- Unstructured overlay
  - Gnutella, Kazaa, Morpheus, Freenet, …
  - Uses flooding

- Structured overlay
  - Exponential network with ring structure
  - DHT (Distributed Hash Table), e.g., Chord, DKS, P2PS

\[ R = N - 1 \text{ (hub)} \]
\[ R = 1 \text{ (others)} \]
\[ H = 1 \]

\[ R = ? \text{ (variable)} \]
\[ H = 1 \ldots 7 \]
(but no guarantee)

\[ R = \log N \]
\[ H = \log N \]
(with guarantee)
Distributed Hash Tables

- Dynamic distribution of a *hash table* onto a set of cooperating nodes

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Algorithms</td>
</tr>
<tr>
<td>9</td>
<td>Routing</td>
</tr>
<tr>
<td>11</td>
<td>DS</td>
</tr>
<tr>
<td>12</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>21</td>
<td>Networks</td>
</tr>
<tr>
<td>22</td>
<td>Grids</td>
</tr>
</tbody>
</table>

- Basic service: *lookup operation*
  - Key resolution from any node
- Each node has a *routing table*
  - Pointers to some other nodes (called “fingers”)
  - Typically, a constant or a logarithmic number of pointers

→ Node D : lookup(9)
Structured overlay networks are based on a ring structure:
- By far the most popular structure, it has many variants and has been extensively studied.

Self organization is done at two levels:
- The ring ensures connectivity: it must always exist despite node joins, leaves, and failures.
- The fingers provide efficient routing: they can be temporarily in an inconsistent state.
The relaxed ring

- False failure suspicions are common on the Internet
  - We do not want to eject the node from the ring when this happens
- The relaxed ring solves this by doing ring maintenance in asynchronous fashion [Mejias et al. 2008]
  - Nodes communicate through message passing
  - For a join, instead of one step involving 3 peers (as in Chord or DKS), we have two steps each with 2 peers → we do not need locking or a periodic stabilization algorithm
- Invariant: Every peer is in the same ring as its successor
Example of a relaxed ring

- It looks like a ring with “bushes” sticking out
- The bushes appear only if there are failure suspicions
  - “Bushiness” increases with failure suspicion rate
- There always exists a perfect ring (in red) as a subset of the relaxed ring
- The relaxed ring is always converging toward a perfect ring
  - The bush structure existing at any time depends on the churn (rate of change of the ring, failures/joins) and the failure suspicion rate
The merge algorithm (1)

- Network partitioning is a common occurrence in realistic networks (such as the Internet)
  - The nodes are partitioned into several groups, with no communication between groups
- With properly designed ring maintenance, each group continues to work as a single structured overlay network
  - But the groups do not communicate, even when the network partition is removed
- The merge algorithm is designed to merge the groups back into a single overlay network [Shafaat et al 2008]
  - Before we designed this algorithm, structured overlay networks would break irreversibly when the network partitioned
The merge algorithm (2)

- The algorithm has two parts
  - Automatic detection of when to merge
    - Each node maintains a passive list of nodes without communication
    - These nodes are pinged periodically
  - Simple ring unification algorithm
    - Assume node $a$ detects node $b$ on another ring
    - Node $a$ calls mlookup($b$) to find $b$’s place in the ring
    - When $b$ is adjacent, then call trymerge($c_{\text{pred}}$, $c_{\text{succ}}$) to insert the node
    - Recursive call to mlookup; stops when mlookup to itself

- Optimized versions of the algorithm use gossip to achieve logarithmic time
SON 1
SON 2

Sept. 2008
Physical analogy
Phase transitions?

- A phase is a set of states of a macroscopic physical system that have relatively uniform chemical composition and physical properties (i.e. density, crystal structure, index of refraction, and so forth).
  - A phase is a region in the parameter space of thermodynamic variables in which the free energy is analytic; between such regions there are abrupt changes in the properties of the system, which correspond to discontinuities in the derivatives of the free energy function.

- Our structured overlay network shows characteristics reminiscent of phases and phase transitions
  - At low failure suspicion rates, the ring is a perfect ring where each node has a fixed set of neighbors (solid phase?)
  - At higher failure suspicion rates, the ring has a bushy structure that is always changing; each node has a varying set of neighbors (liquid phase?)
  - At yet higher failure suspicion rates, the ring degenerates into several disconnected rings, and at highest failure suspicion (failed communication), each node is a ring of size 1 (gaseous phase?)
The relaxed ring has (at least) three phases

- We are studying its behavior to understand how the ring reacts to external parameters (including phase transitions)
Some remarks

- Analytic study of Chord shows three phases with transitions as network delays increase [Krishnamurthy and Ardelius, 2008]
  - Chord is an idealized structured overlay network with simple algorithms
  - Three phases: (1) a region of efficient lookup, (2) a region of inefficient lookup (long fingers are dead), (3) a region of disconnected ring
  - The inefficient lookup is due to a positive feedback effect: incorrect fingers lead to longer lookup, which at some point cannot be fixed since lookup is too slow to allow fixing the fingers (the network has changed in the meanwhile)

- In our own situation, things are not so simple
  - Input network parameters: size $n$ (number of nodes), successor list redundancy $f$ (small integer), failure suspicion rate $r$ ($0 \leq r \leq 1$), churn $c$ ($0 \leq c \leq 1$, rate of node turnover)
  - $n$ and $f$ are imposed by system structure, $r$ and $c$ are imposed by environment
  - Output network parameters: perfection $p$ ($0 \leq p \leq 1$), entropy $s$ ($0 \leq s \leq n \cdot \ln(n)$), lookup efficiency $e$ ($e \geq 1$, as compared to best fingers), lookup inconsistency rate $i$ ($0 \leq i \leq 1$)
Simulation study

- We are currently performing simulations to study the behavior of practical structured overlay networks
  - Chord: simplest system, uses locking and periodic stabilization
  - P2PS: relaxed ring with merge algorithm, uses no locking
- This is work in progress in the SELFMAN project
Design methodology

- Design software systems as a set of interacting feedback loops
  - Each feedback loop controls part of the system
  - The feedback loops interact to manage the overall system
  - Phase transitions will occur naturally as a result of external parameters
- Design software systems so that phase transitions are reversible
  - They will “self heal” when the external stress causing the transition is removed
  - This may require the design of specialized algorithms (e.g., structured overlay network with merge algorithm)
- What design methodology should we use?
  - We need to design for a desired system behavior
  - Analytic study is prohibitive and simulation is only indicative
- Research agenda: create a methodology usable in practical software development
  - First approach (intuitive): study existing systems and derive design rules
  - Second approach (rigorous): prove correctness of design rules by using translations to process calculi

Sept. 2008
Some conclusions

- To increase robustness and adaptiveness, software can be designed as interacting feedback loops
  - By analogy from the physical and biological sciences
- Phase transitions are a natural consequence of feedback loop architectures
  - For robustness, we need to design reversible phase transitions
- We need a methodology for designing these systems
  - How to design a feedback loop structure to achieve desired robustness
  - How to achieve the desired phases and phase transitions
  - There is a research agenda here