Footloose: A Case for Physical Eventual Consistency and Selective Conflict Resolution

Justin Mazzola Paluska
David Saff, Tom Yeh, Kathryn Chen

MIT CSAIL
Today’s Situation

- Data is scattered throughout devices:
  - All of my phone numbers on my cell phone
  - Some other contact information on my PDA
  - Still more on my laptop
- But no way to manage data
Device Characteristics

- A single primary user
- Some memory
- A wireless communications medium
- Shared contexts

- What can we do with these resources?
Effective Use of Communication Pathways
Formal Requirements

- Distribution of heterogeneous data for increased availability
- Optimistic writes on all devices with application-level conflict resolution
- Automatic management of replicas
- On whatever network is available
New Assumptions

- Mostly disconnected operation on non-Internet networks
- Applications may run on devices that may never directly talk
- Devices only understand and can resolve conflicts for a few data types
- Devices have finite storage capabilities
New Solutions

- Physical Eventual Consistency
  - Use a pervasive device’s location to enhance consistency
- Selective Conflict Resolution
Physical Eventual Consistency
Physical Eventual Consistency

- Weak eventual consistency
- “Sneaker net” approach to data transfer
- The device with the most updates should be closest to the user.
Selective Conflict Resolution

- Two classifications:
  - Smart – can resolve conflicts
  - Dumb – cannot resolve conflicts

- All devices can move all data

- Separate conflict transfer from conflict resolution
Footloose

- Shared data store for pervasive applications
  - Guarantees “no lost updates”
  - Automatic management and routing of shared data.
  - Application-level device-distributed conflict resolution
Footloose Architecture

User Application
Register types that will be stored and shared.

Footloose Store
Application interface and storage facility.

Footloose Protocol Daemon
Device interface and consistency maintainer.

Any possible network connection
The Footloose Store

- Mutable mapping for applications
  - RecordID maps to a set of UpdateEvents
  - Only valid UpdateEvents get returned to apps.
- Applications make updates, not writes.
- The FLS stores only UpdateEvents
- List of UpdateEvents for the FPD

- Easy conflict detection!
- Transparent Transport of Conflicts!
Enabling Automatic Management

\[
\{ \text{NULL} \rightarrow \text{UE}_1: 555-1000, \\
\text{UE}_1 \rightarrow \text{UE}_2: 867-5309 \} = \{ \text{NULL} \rightarrow \text{UE}_1: 555-1000, \\
\text{UE}_1 \rightarrow \text{UE}_2: 867-5309 \}
\]
The Footloose Protocol Daemon

- Maintains “shared knowledge” about various agents on all devices
- Device interests
- Manages Physical Communication
- Data routes

Garbage Collection & Purging

FPD

{UpdateEvent, StatusVector} → {UpdateEvent, StatusVector}

Foreign FPD

{UpdateEvent, StatusVector}, {Device Interests}
Effective Use of Communication Pathways

laptop:
Interest in = {laptop, work-comp}

{NULL→UE₁: , ...

work-comp:
Interest in = {laptop, work-comp}

{NULL→UE₁: , ...

cell:
Interest in = {laptop, work-comp}

{NULL→UE₁: , }
Implementation

- FLS and FPD built in Java
- Two applications:
  - Wishlist Application
  - Phone Number Database
    - 25 lines of Footloose-dependent code
- Simulation Framework
Evaluation

The graph shows the number of updates required as a function of the number of devices for different network topologies: Clique, Line, Star, Tree, and Ring. The number of updates increases significantly with the number of devices, indicating higher communication overhead in more complex networks.

Key:
- **Clique**
- **Line**
- **Star**
- **Tree**
- **Ring**

**Axes:**
- **Y-axis:** Number of Updates
- **X-axis:** Number of Devices
Evaluation

The diagram shows the number of syncs required for different buffer sizes with varying numbers of devices.

- **3 devices**: The number of syncs remains relatively constant across different buffer sizes.
- **4 devices**: A slight increase in syncs is observed as the buffer size increases.
- **5 devices**: A noticeable increase in syncs with larger buffer sizes.

The x-axis represents the buffer size, while the y-axis shows the number of syncs. The graph indicates that as the buffer size increases, the number of syncs required also increases, especially when dealing with a larger number of devices.
Design Evaluation and Future Work

- Need better support for complex “directory-like” types
- Support for large updates
- Variable Number of Devices
- Security Framework
- User Study
Further Information

Justin Mazzola Paluska
jmp@mit.edu
http://web.mit.edu/jmp/www/