Porcupine

Robert Grimm
New York University
Final presentations on Tuesday 5/10, 2-5pm, 613 WWH

- Conference format: One slot per group
  - 20 minutes for your talk
    - If you can, include a live demo
  - 10 minutes for questions
    - Your audience are your class mates and me
Altogether Now: The Three Questions

- What is the problem?
- What is new or different or notable?
- What are the contributions and limitations?
Porcupine from 1,000 Miles

- Cluster-based email service
  - Real need (think Hotmail, Yahoo! Mail)
  - Write intensive workload (unlike many web sites)
  - Loose consistency requirements (we can be clever)

- Goals, techniques, overarching principle
  - Availability, manageability, performance
  - Replication, automatic reconfiguration, dynamic scheduling
  - Functional homogeneity
    - “Any node can perform any task”
(Replicated) State in Porcupine

- **Hard state**
  - The actual information of the service
  - Stored in stable storage

- **Soft state**
  - Information that can be reconstructed from hard state
  - Mostly stored on a single node
  - However, directories referencing other soft state are replicated across all nodes
    - User map
    - Cluster membership list
Key Data Structures

- Mailbox fragment
  - Part of user’s mail, unit of replication
  - Hard state

- Mail map
  - Mapping between users and mailbox fragments
  - Soft state

- User profile database
  - Client descriptions
  - Partitioned, replicated
  - Hard state
Key Data Structures (cont.)

- **User profile soft state**
  - Cache of user profile database

- **User map**
  - Mapping between Hash(user) and node managing user’s profile soft state and part of mail map
  - Soft state replicated across all nodes

- **Cluster membership list**
  - View of currently functioning nodes
  - Soft state replicated across all nodes
The Functional View: Porcupine Modules

- Front end
  - Proxies
- Middle-tier
  - Load balancer
  - Membership manager
  - RPC manager
- Backend
  - User manager
  - Replication manager
  - Mailbox manager
  - User DB manager
An Example Configuration

- **Soft state**: One node per user
  - Typically replicated for availability
- **Hard state**: One or more nodes per user
Mail Delivery

- MTA contacts any Porcupine node using SMTP
- SMTP proxy
  - Hashes user name to locate managing node
  - Retrieves mail map from user manager
  - Asks load manager to choose best node
  - Forwards message to mailbox manager
- Mailbox manager updates mail map if necessary
Mail Retrieval

- MUA contacts any Porcupine node w/POP/IMAP
- POP/IMAP proxy
  - Authenticates user through user manager
  - Retrieves digest information from all mailbox managers
  - Fetches message from appropriate node(s)
  - Forwards deletion requests to appropriate node(s)
Decoupling has (potential) advantages
- Dynamic balancing of mail delivery
  - Any node can manage user information
  - Any node can store mail messages
- Fault tolerance
  - Nodes storing mail need not be available
- No human intervention
  - On node addition, failure, and removal

Key tension
- Distribution necessary for load balancing and availability
- But too much distribution complicates delivery & retrieval
This Looks Complicated... Any Alternatives?

- Static partitioning (IBM, HP)
  - Hard to manage & scale
- Distributed file system (UCB xFS, Frangipani)
  - Semantics ill-suited to workload
- Cluster-based OS’s (IBM, Sun)
  - Pretty expensive, limited scalability
- Big iron (IBM)
  - Very expensive, hard scalability limit
Self-Management

- Cluster membership protocol
  - Node detects change and suggests new epoch
  - Other nodes accept epoch
  - Coordinator confirms membership and epoch

- User map updated through membership protocol
  - Each node notifies coordinator of its part of user map
  - Coordinator redistributes buckets, minimizes changes
Self-Management (cont.)

- Rest of soft state updated through two-step, distributed and unsynchronized process
  - Each node calculates difference between user maps
  - Each node sends new manager soft state for its hard state
    - Identify mailbox fragments
    - Send part of user profile database (only node with highest IP addr.)

- Cost per node is constant, independent of cluster size
  - Dominated by discovery of mailbox fragments
    - Proportional to number of reassignments to user map
    - Inversely proportional to cluster size
  - Number of failures proportional to cluster size
Self-Management in Action: What Happens If...

- Mailbox fragment node fails after message stored in new fragment?
- Mailbox fragment node fails after last message deleted from fragment?
- User manager node fails after message stored in new fragment?
- User manager node fails after last message deleted from fragment?

See page 312
Replication Properties

- Update anywhere
  - All nodes are created equal
- Eventual consistency
  - Expose inconsistencies for (hopefully) short times
- Total update
  - Always change the entire object
- Lock free
  - We don’t use distributed locks
- Ordered by loosely synchronized clocks
  - Use wall clock time to order competing updates
Implications of Replication Properties

- Content may change in surprising ways
  - The same email message may be delivered twice
  - A deleted email message may appear again
- There may be different views of the same data
  - Multiple agents running for the same user may see different mailbox contents
- The same is true for the user database
  - A user may only exist on some machines
  - Passwords may be different
Replication in Action

- Based on update log
  - \(<\text{timestamp}, \text{objectID}, \text{target-nodes}, \text{remaining-nodes}>\>

- First replica coordinates update process
  - Pushes object and log entry to each remaining replica
  - Confirms process once all replicas have ack-ed update
    - Update can be retired from log after waiting period to detect out-of-order delivery
      - Sum of maximum clock skew and max network packet lifetime

- Each replica logs all updates
  - Optimization: Last replica need not log update
    - In practice with 2 replicas, only coordinator logs update
Failures during Replication

- Coordinator fails before responding to proxy
  - Proxy creates a new object, selects a new set of replicas, and tries again
- Coordinator fails before update is applied to all replicas
  - Some replica takes over and pushes the update along
- Issue: Update log may become really large
  - Updates only remain in log for up to a week
  - Nodes that are down for longer rejoin as “new”
Dynamic Load Balancing

- **Goals**
  - Make decisions at message delivery granularity
  - Support heterogeneous cluster
  - Be automatic, avoid magic constants
  - Resolve tension between load and affinity

- **Basic implementation**
  - Each proxy makes local load-balancing decisions
  - Load information
    - Piggy-backed through RPC exchanges
    - Systematically collected through a virtual ring
    - Expressed as number of pending RPCs that might access disk
What about Affinity?

- Load alone tends to distribute mailboxes across many nodes
- Therefore, we need to limit *spread*
  - Soft upper bound on nodes with user’s mail
  - Not only upper, but also lower limit
    - Add random nodes when user’s mail is not spread enough
System Evaluation

- **Performance**
  - Single-node
  - Scalability over nodes
  - Comparison to statically partitioned cluster

- **Availability**
  - Cost of replication and reconfiguration

- **Manageability**
  - Recovery speed
  - Effect of incremental hardware improvements
  - Effect of load balancing on highly skewed workloads
Basic Experimental Setup

- 30 Linux-based PCs
  - 6 times: 200 MHz, 64 MB, 4 GB SCSI
  - 8 times: 300 MHz, 128 MB, 4 GB IDE
  - 16 times: 350 MHz, 128 MB, 8 GB IDE
- 1 Gb/s Ethernet
  - Not switched
Synthetic Workload

- Modeled on departmental mail server
- Message size
  - Mean: 4.7 KB
  - Heavy tail up to 1 MB
- Transaction frequency
  - 90% SMTP, user chosen according to Zipf distribution
  - 10% POP (!)
- User population: 160,000 * |nodes| = 5 million
- Cluster saturated
  - Message deliveries counted
- Replication increases disk writes threefold
  - Once for each replica
  - Once for the coordinator’s log
Are Disks the Bottleneck?

- Single node with one disk
  - CPU utilization with replication is 12%
  - Disk utilization with replication is 75%

- Single 300 MHz node
  - 105 messages/sec for 1 IDE + 2 SCSI disks
  - 23 messages/sec for 1 IDE disk
What If We Had Infinitely Fast Disks?

- Performance improves 6-fold over real system
  - CPU becomes the bottleneck
- However, network also has limited capacity
  - 6500 messages/sec for 4.7 KB message w/o replication
Effects of Load Balancing

- What is the trade-off between static, dynamic, & random?
Heterogeneous Configurations and Failure Recovery w/Replication

Fig. 13. Reconfiguration timeline with replication.
Discussion

- Functional homogeneity certainly is elegant
  - But what are the limitations?
- What drives the implementation?
  - Exploit domain-specific knowledge
    - Membership protocol combined with user map updates
    - Optimistic replication to provide eventual consistency
- Is the system complete?
  - What happens if nodes are continuously replaced?
- What other applications?
  - News, bulletin boards, calendaring for sure
    - But, what else?
What Do You Think?