Clusters

Or: How to replace Big Iron with PCs

Robert Grimm
New York University
Before We Dive into Clusters...

- Assignment 2: Better Support for HTTP/1.1
  - Due 10/7/03
  - You might want to wait with coding until the next release of Munin later this week
- Groups
Clusters and Giant-Scale Services
Basic idea
- Use off-the-shelf PCs to run highly-scalable services

Challenges
- Want availability, performance, manageability
- But achieving all three properties is not trivial
  - Consistency of distributed state becomes a major concern
Supporting Giant-Scale Services
Take 1: MPPs (Big Iron)

- Workstation-class nodes connected by dedicated, low-latency network
  - Issues
    - Engineering lag time
    - Costs (e.g., special hardware and software)
    - Limited versatility (think interactive performance)
  - Advantages
    - Communication performance
      - Network interface close to processor
    - Global system view
      - One task, not a collection of processes
Supporting Giant-Scale Services
Take 2: Clusters to the Rescue

- “Networks of Workstations” (UC Berkeley)
  - Switched local-area networks
  - Powerful workstations
  - I/O bottleneck
Anatomy of a Cluster-Based Service

- IP network
- Load manager
  - DNS round robin
  - Layer-4/7 switches
- Servers
  - Web
  - Business logic
- Data store
- Internal network
  - Backplane
Availability Metrics

- Uptime, typically expressed in “nines”
  - E.g., four nines means 0.9999 uptime, or less than 60 seconds downtime per week
- Mean-time-between failures (MTBF) and mean-time-to-repair (MTTR)
  - uptime = (MTBF – MTTR) / MTBF
- Yield and harvest
  - yield = queries completed / queries offered
  - harvest = data available / complete data
One More Metric: DQ

- Data per query * queries per second \( \rightarrow \) constant
  - Overall capacity determined by physical bottleneck
    - Network bandwidth
    - Disk performance

- DQ is measurable and tunable
  - Depends on workload, hardware, software, data
  - Quantifies relative impact of faults, database size, hardware and software upgrades
  - Normally scales linearly with number of nodes
    - Use small cluster to predict impact on production cluster
Replication vs. Partitioning under Faults

- Impact on harvest, yield, DQ
  - Replication maintains harvest, but reduces yield
  - Partitioning maintains yield, but reduces harvest
  - Both lose DQ

- Load redirection problem for replication
  - DQ bottleneck is the real cost, not disk space
    - Hence: $n/(n-k)$ overload for $k$ failures and $n$ nodes

- Still want replication, at least for key data
  - Only way to achieve high throughput
  - Better control over harvest, easier to recover, grow
Graceful Degradation

- Can we avoid saturation? Not really
  - Decide how saturation affects uptime, harvest, quality
  - Consider admission control to implement your policy

- Examples for graceful degradation
  - Cost-based AC
    - Refuse queries you estimated to be expensive
  - Priority- or value-based AC
    - Give preference to certain queries
  - Reduced data freshness
    - Decrease work per query
Online Evolution

- Internet-time implies constant change
  - Need acceptable quality
    - Meet target MTBF, low MTTR, no cascading failures
- Three approaches to managing upgrades
  - Fast reboot: Cluster at a time
    - Minimize yield impact
  - Rolling upgrade: Node at a time
    - Versions must be compatible
  - Big flip: Half the cluster at a time
    - Reserved for complex changes
  - Either way: use staging area, be prepared to revert
Eric Brewer’s Lessons

- Get the basics right
- Decide on your availability metrics
- Focus on MTTR at least as much as MTBF
- Understand load redirection during faults
- Graceful degradation is a critical for availability
- Use DQ analysis on all upgrades
- Automate upgrades as much as possible
A Closer Look at One Cluster-Based Service
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
Key Data Structures

- Mailbox fragment
  - Part of user’s mail on single node
  - Unit of replication
  - Hard state

- Mail map
  - Mapping between users and nodes
  - 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 mail map
  - Soft state replicated across all nodes
- Cluster membership list
  - View of currently functioning nodes
  - Soft state replicated across all nodes
Data Structure Managers

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

[Diagram showing two nodes, A and B, with user profiles and mailbox fragments.]
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 chose 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)
So Far, So Good

- 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
    - Node addition, failure, and removal

- Key tension
  - Degree of distribution impacts mail delivery & retrieval
  - Therefore, need to limit spread of user’s mail
This Looks Complicated... Any Alternatives?

- Big iron
  - IBM
- Cluster-based OS
  - IBM, Sun
- Distributed file system
  - Berkeley’s xFS
  - HP SRC’s Frangipani
- Static partitioning
  - IBM, HP
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 manager soft state for its hard state
    - Identify mailbox fragments
    - Send part of user profile database

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

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

- See page 312
Replication and Availability

- Replication properties
  - Update anywhere
    - All nodes are created equal
  - Eventual consistency
    - Expose inconsistencies for (hopefully) short times
  - Total object
    - 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
  - <timestamp, objectID, target-nodes, 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

- 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
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 sized messages
Effects of Load Balancing under Different Policies

![Graph showing the effects of load balancing under different policies.](image)
Effects of Load Balancing (cont.)

- Evaluate static spread, random, dynamic spread
  - Random not subject to skew but ignores affinity
  - Static spread can lead to unbalanced load
  - Dynamic balances the two
    - As long as there is more than one node in the spread
Adaptation to Heterogeneous Configuration
Failure Recovery

The diagram illustrates the messages per second over time for different failure scenarios. The x-axis represents the timeline in seconds, ranging from 0 to 800. The y-axis shows the number of messages per second, ranging from 100 to 300.

Key events include:
- **Nodes fail**: The initial point where nodes begin to fail.
- **New membership determined**: The point where a new membership is determined post-failures.
- **Nodes recover**: The point where nodes recover from the failure.
- **New membership determined**: The second point where a new membership is determined post-recovery.

The graph shows three distinct lines indicating the number of messages per second under different conditions:
- **Solid line**: No failure.
- **Dashed line**: One failure.
- **Dash-dotted line**: Three failures.

The graph demonstrates how traffic density and latency are affected by these failures and recoveries.
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
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 about e-commerce?