Internet Server Clusters

Using Clusters for Scalable Services

Clusters are a common vehicle for improving scalability and availability at a single service site in the network.

- incremental scalability
  - just wheel in another box...
- excellent price/performance
  - high-end PCs are commodities: high-volume, low margins
- fault-tolerance
  - "simply a matter of software"
- high-speed cluster interconnects are on the market
  - SANs + Gigabit Ethernet...
- cluster nodes can coordinate to serve requests w/ low latency
  - "shared nothing"

The Porcupine Wheel

Replication

availability

scale

Functional
homogeneity

Dynamic transaction scheduling

Automatic reconfiguration

Porcupine Replication: Overview

To add/delete/modify a message:
- Find and update any replica of the mailbox fragment. *Do whatever it takes*: make a new fragment if necessary. *pick a new replica if chosen replica does not respond.*
- Replica asynchronously transmits updates to other fragment replicas. *continuous reconciling of replica states*
- Log/force pending update state, and target nodes to receive update. *on recovery, continue transmitting updates where you left off*
- Order updates by *loosely synchronized/physical clocks*. Clock skew should be less than the inter-arrival gap for a sequence of order-dependent requests. *use nodeID to break ties.*
- How many node failures can Porcupine survive? *What happens if nodes fail "forever"?*

Yasushi’s Slides

Yasushi’s slides can be found on his web site at HP. http://www.hpl.hp.com/personal/Yasushi_Saito/
I used his job talk slides with a few of my own mixed in, which follow.

Porcupine: A Highly Available Cluster-based Mail Service

Yasushi Saito
Brian Bershad
Hank Levy

http://porcupine.cs.washington.edu/
University of Washington
Department of Computer Science and Engineering,
Seattle, WA
Key Points

- COTS/NOW/ROSE off-the-shelf
- Shared-nothing architecture (vs. shared disk)
- Functionally homogeneous (anything anywhere)
- Hashing with balanced bucket assignment to nodes
- ROWA replication with load-balancing reads
  - Read one write all
- Soft state vs. hard state: minimize hard state
- Leverage weak consistency: “ACID vs. BASE”
- Idempotent updates and total ordering
  - Loosely synchronized clocks
- Operation logging/restart
- Spread and affinity

How Do Computers Fail?

Porcupine’s failure assumptions
Large clusters are unreliable.

Assumption: live nodes respond correctly in bounded time most of the time.

- Network can partition
- Nodes can become very slow temporarily.
- Nodes can fail (and may never recover).
- Byzantine failures excluded.

Taming the Internet Service Construction Beast

Persistent, Cluster-based Distributed Data Structures

(in Java!)

Steve D. Gribble
gribble@cs.berkeley.edu

Ninja Research Group
(http://ninja.cs.berkeley.edu)
The University of California at Berkeley
Computer Science Division

Gribble’s Slides

Steve Gribble’s slides can be found on his web site at UW.

Go to “selected talks” and for the slides on DDS.
I actually used his job talk slides with a few of my own mixed in on
the basics of two-phase commit, which follow.
It is important to understand the similarities/differences between
Porcupine and DDS, and how they flow from the failure
assumptions and application assumptions for each project.

Committing Distributed Transactions

Transactions may touch data stored at more than one site.

Each site commits (i.e., logs) its updates independently.

Problem: any site may fail while a commit is in progress, but after
updates have been logged at another site.

- An action could “partly commit”, violating atomicity.
- Basic problem: individual sites cannot unilaterally choose to abort
  without notifying other sites.
  - “Log locally, commit globally.”

Two-Phase Commit (2PC)

Solution: all participating sites must agree on whether or not each
action has committed.

- Phase 1: The sites vote on whether or not to commit.
  - precommit: Each site prepares to commit by logging its updates
    before voting “yes” (and enters prepared phase).
  - commit: Commit all sites voted to commit.

- Phase 2: The central transaction coordinator gathers the votes.
  If any site votes “no”, the transaction is aborted.
  Else, coordinator writes the commit record to its log.
  Coordinator notifies participants of the outcome.

Note: one server => no 2PC is needed, even with multiple clients.
The 2PC Protocol

1. Tx requests commit, by notifying coordinator (C)
   - C must know the list of participating sites.
2. Coordinator C requests each participant (P) to prepare.
3. Participants validate, prepare, and vote.
   - Each P validates the request, logs validated updates locally, and responds to C with its vote to commit or abort.
   - If P votes to commit, Tx is said to be “prepared” at P.
4. Coordinator commits.
   - If all P votes are unanimous to commit, C writes a commit record to its log, and reports “success” for commit request. Else abort.
5. Coordinator notifies participants.
   - C asynchronously notifies each P of the outcome for Tx.
   - Each P logs outcome locally and releases any resources held for Tx.

Handling Failures in 2PC

1. A participant P fails before preparing.
   - Either P recovers and votes to abort, or C times out and aborts.
2. Each P votes to commit, but C fails before committing.
   - Participants wait until C recovers and notifies them of the decision to abort. The outcome is uncertain until C recovers.
3. P or C fails during phase 2, after the outcome is determined.
   - Carry out the decision by reinitiating the protocol on recovery. Again, if C fails, the outcome is uncertain until C recovers.

More Slides

The following are slides on “other” perspectives on Internet server clusters. We did not cover them in class this year, but I leave them to add some context for the work we did discuss.

Clusters: A Broader View

MSCS (“Wolfpack”) is designed as basic infrastructure for commercial applications on clusters.

- “A cluster service is a package of fault-tolerance primitives.”
- Service handles startup, resource migration, failover, restart.
- But: apps may need to be “cluster-aware”.
  Apps must participate in recovery of their internal state.
  Use facilities for logging, checkpointing, replication, etc.
- Service and node OS supports uniform naming and virtual environments.
  Preserve continuity of access to migrated resources.
  Preserve continuity of the environment for migrated resources.

Wolfpack: Resources

- The components of a cluster are nodes and resources.
  - Shared nothing: each resource is owned by exactly one node.
- Resources may be physical or logical.
  - Disks, servers, databases, mailbox fragments, IP addresses,...
- Resources have types, attributes, and expected behavior.
- (Logical) resources are aggregated in resource groups.
  - Each resource is assigned to at most one group.
- Some resources/groups depend on other resources/groups.
  - Admin-installed registry lists resources and dependency tree.
- Resources can fail.
  - Cluster service/resource managers detect failures.

Fault-Tolerant Systems: The Big Picture

- messaging system
- file/storage system
- database
- mail service
- cluster service
- application service
- replication
- logging
- checkpointing
- voting
- RAID parity
- checksum
- redundancy
- failover
- EAC
- retransmission
- dependencies
- redundancy at any/each/every level
- what failure semantics to the level above?
**Wolfpack: Resource Placement and Migration**

The cluster service detects component failures and responds by restarting resources or migrating resource groups.

- Restart resource in place if possible...
- ...else find another appropriate node and migrate/restart.

Ideally, migration/restart/failover is transparent.

- Logical resources (processes) execute in virtual environments.
- uniform name space for files, registry, OS objects (NT mods)
- Node physical clocks are loosely synchronized, with clock drift less than minimal time for recovery/migration/restart.
- Guarantees migrated resource sees monotonically increasing clocks
- Route resource requests to the node hosting the resource.
- Is the failure visible to other resources that depend on the resource?

---

**TACC**

Vision: deliver “the content you want” by viewing HTML content as a dynamic, mutable medium.

1. **Transform** Internet content according to:
   - network and client needs/limitations
     - e.g., on-the-fly compression/distillation [ASPLOS96], packaging Web pages for PalmPilots, encryption, etc.
     - directed by user profile database
   2. **Aggregate** content from different back-end services or resources.
   3. **Cache** content to reduce cost/latency of delivery.
   4. **Customize** (see Transform)

---

**SNS/TACC Philosophy**

1. Specify services by plugging generic programs into the TACC framework, and compose them as needed.
   - sort of like CGI with pipes
   - run by long-lived worker processes that serve request queues
   - allows multiple languages, etc.
   - ACID vs. BASE
   - Worker processes in the TACC framework are loosely coordinated, independent, and stateless.

2. Handle bursts with designated overflow pool of machines.

---

**[Fox/Brewer]: SNS, TACC, and All That**

[Fox/Brewer97] proposes a cluster-based reusable software infrastructure for scalable network services (“SNS”), such as:

- **TranSend**: scalable, active proxy middleware for the Web think of it as a dial-up ISP in a box, in use at Berkeley distills/transforms pages based on user request profiles
- **Inktomi/HotBot search engine**
  - core technology for Inktomi Inc., today with $15B market cap.
  - “bringing parallel computing technology to the Internet”

Potential services are based on Transformation, Aggregation, Caching, and Customization (TACC), built above SNS.
**(Worker) Ignorance Is Bliss**

What workers don’t need to know

- Data sources/sinks
- User customization (key/value pairs)
- Access to cache
- Communication with other workers by name

Common case: stateless workers

C, Perl, Java supported

- Recompilation often unnecessary
- Useful tasks possible in <10 lines of (buggy) Perl

---

**Questions**

1. What are the research contributions of the paper?
   - System architecture decouples SNS concerns from content
   - TACC programming model composes stateless worker modules
   - Validation using two real services, with measurements
   - How is this different from clusters for parallel computing?

2. How is this different from clusters for parallel computing?

3. What are the barriers to scale in SNS/TACC?

4. How are requests distributed to caches, FEs, workers?

5. What can we learn from the quantitative results?

6. What about services that allow client requests to update shared data?
   - e.g., message boards, calendars, mail,

---

**SNS/TACC Functional Issues**

1. What about fault-tolerance?
   - Service restrictions allow simple, low-cost mechanisms.
     - Primary/backup process replication is not necessary with BASE model and stateless workers.
   - Uses a process-peer approach to restart failed processes.
     - Processes monitor each other’s health and restart if necessary.
     - Workers and manager find each other with “beacons” on well-known ports.

2. Load balancing?
   - Manager gathers load info and distributes to front-ends.
   - How are incoming requests distributed to front-ends?