Fault Tolerance for Highly Available Internet Services: Concept, Approaches, and Issues

By Narjess Ayari, Denis Barbaron, Laurent Lefevre and Pascale Primet

Presented by Mingyu Liu
1. Introduction
   - FT Concepts & Challenges

2. Fault Models & Failure Detection
   - Approaches & Issues

3. Service Replications
   - Concepts, Approaches & Issues

4. Failure Recovery
   - Network, Transport, Session/Application Level Failovers

5. Conclusion
Fault Tolerance Framework

- FT Frameworks uses **Resource Redundancy** to Ensure Availability

- Two Concepts
  - Fault Detection
  - Fault Recovery

- Three Challenges
  - Resource Consumption
  - Strength of Fault Tolerance
  - Performance

Redundancy in Cluster-based Architecture

- Two Redundancy Scenarios
  - **Passive Scenario**
  - **Active Scenario**

Fault Models

Fault Types and Models

- **Fault Types**
  - Client-side fault
    - concerns the client device
  - Network-side fault
    - includes corruption, delay, reordering, duplication, and loss of packets
  - Server-side fault
    - results in the silence or malfunctioning of the processing server

- **Fault Models**
  - Byzantine fault
    - occurs arbitrarily and maliciously, causing the system to behave incorrectly
  - Fail-stop fault
    - has a deterministic impact on a subsystem component, causing it die silently
    - inactive during failure
Failure Detection Approaches

- **Requirement**
  - It should detect failures as soon as they occur so that the framework can quickly trigger the failure recovery procedure.
  - It must be robust enough to ensure that only one error-free instance of the service is running at once.

- **Heartbeat Monitoring**
  - Based on the explicit and periodic exchange of heartbeat messages between replicas.

Heartbeat Monitoring

- Two monitoring types:

- *Push-based heartbeat monitoring*

```plaintext
The monitor process
function failure_detector(Host h)
    On receive {Heartbeat_Hello} from h
        return up;
    After n*δ
        return crashed;

The monitored process
procedure Availability_announce()
    Forever
        Send {Heartbeat_Hello} to the monitor
        Wait δ
```

- *Pull-based heartbeat monitoring*

```plaintext
The monitor process
function failure_detector(Host h)
    Send {Heartbeat_Hello} to the receiver
    Wait δ
    On receive {Heartbeat_Reply}
        return up;
    After n*δ
        return crashed;

The monitored process
procedure Availability_announce()
    Forever
        On receive {Heartbeat_Hello}
            Send {Heartbeat_Reply} to the monitor
```

Problem with Heartbeat Monitoring

- Heartbeat monitoring is generally used to detect a node or link failure
- Failure could occur at a smaller level
  - such as at process level

Solution

- **Watchdog timer** is an inexpensive solution
  - process being monitored must reset a timer before it expires
  - otherwise, it is assumed to have failed
- Problems with Waterdog
  - only deterministic runtime process can be monitored
  - partially failed process can still reset the timer
Replication Concept

- Recovery of a service by replicating its related states
- When failure occurs, the traffic is taken over by an elected backup node

Requirements

- Transparency
  - Needs to achieve client-side transparent failover, already established sessions need to be recovered in case of failure
- Overhead
  - Measured by the cost of replication process during failure-free period
- Consistency
  - Needs replicas to maintain the same view of the replicated states

Replication Approaches

- Leader/follower
- Active Replication
- Checkpointing
- Message Logging
- Hybrid Approach
Replication

Leader/follower Approach

- **Idea**
  - Let a replica (leader) perform action first;
  - Then leader notifies followers the results;
  - Replicas update their state.

- **Evaluation**
  - Performs well with read-only files
  - Not appropriate for processes modifying files concurrently
  - Performs poorly when large volumes of info involved
Replication

Active Approach

- Idea
  - All nodes to receive and concurrently process the offered network traffic
  - Its objective is to ensure all replicas maintain same state and guarantee only one server replies to client

- Evaluation
  - Leader does not need to forward data to followers
  - Further processing is required to ensure consistency
    - Atomic Multicast Protocol
    - Intermediate Gateway or Proxy
    - etc.

Diagram:
- Input processing
- Primary
- Backup
- Output consolidation
- Atomic multicast
- Atomic broadcast
- Traffic sniffing
- Etc.
Checkpointing Approach

- **Idea**
  - State is periodically copied either to standby servers or to a stable storage
  - **Incremental Checkpointing** checkpoints each time change occurs
  - **Time-line Checkpointing** checkpoints state periodically

- **Evaluation**
  - Aggressive approach has high cost and adds latency
  - Time-line approach’s time-to-check value affects overhead and number of rollback operations
Replication

Message Logging Approach

- Idea
  - To store or log all the messages delivered to the primary server on stable storage or a replica
  - Dependency-based Logging flushes the log space once full
  - Optimistic Logging flushes periodically or at a given threshold

- Evaluation
  - Recover time takes longer than checkpointing approach
Active replication and Message logging need server to be deterministic
Active replication has the best recovery time
Message logging needs longest recovery time

<table>
<thead>
<tr>
<th></th>
<th>Active replication</th>
<th>Message logging</th>
<th>Checkpointing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Resource usage</strong></td>
<td>Requires a dedicated backup</td>
<td>Requires an idle backup</td>
<td>Frequent checkpoint is costly</td>
</tr>
<tr>
<td><strong>State preservation frequency</strong></td>
<td>States are created on the fly</td>
<td>Connection-level messages are logged</td>
<td>With every state change, etc.</td>
</tr>
<tr>
<td><strong>Recovery time</strong></td>
<td>Short</td>
<td>Long (message log replay)</td>
<td>Less than the time required in the logging scheme</td>
</tr>
<tr>
<td><strong>Failure-free overhead</strong></td>
<td>Active replication scheme dependent</td>
<td>Additional delay</td>
<td>The commit delay overhead</td>
</tr>
<tr>
<td><strong>Nondeterminism handling</strong></td>
<td>Must be handled by the active replication method</td>
<td>Issue for the connection and application level</td>
<td>Undefined</td>
</tr>
<tr>
<td><strong>Need for message interception</strong></td>
<td>Depends on the primary/backup topology</td>
<td>Depends on the primary/backup topology</td>
<td>Depends on the primary/backup topology</td>
</tr>
</tbody>
</table>
Failure recovery is followed by detection

- Its objective is to increase both availability and reliability
- **Network identity takeover** is the first step
- Further steps needed to meet reliability requirement
  - Transport-level failover
  - Session/Application level failover
Failover

Network-level Failover

- Idea
  - Provide replicas the means to take over the network identity of the legitimate processing server if it fails.
  - It provides an acceptable level of service availability

- Approaches
  - Link Aggregation Protocol
    - allows the use of multiple Ethernet network interfaces or links in parallel
  - ARP-Spoofing-based network Identify Takeover
    - backup node takes over the virtual IP by flooding gratuitous ARP message
  - Virtual Router Redundancy Protocol
    - virtual router abstracts a cluster of routers servicing hosts in the same network
  - Static NAT-based IP takeover
    - traffic first offered to the entry point before assigning to a server
Failover

Transport-level failover

☐ Idea
  ▪ Should the primary server fail, the already established flow is taken over by an elected backup while avoiding its interruption.

☐ Approaches
  ▪ FT-TCP
  ▪ Transparent Connection Failover
  ▪ ST-TCP

Session/Application Level Failover

☐ Idea
  ▪ Require the elected replica to failback each associated state

☐ Approaches
  ▪ Synchronize the primary node’s system call at each replica
  ▪ Identify nondeterministic behaviour at the application level and synchronizing at those point
  ▪ Use checkpointing to save the primary’s application level state
This paper provides a comprehensive overview of the building blocks of fault tolerance frameworks.

- Fault model and failure detection approaches
  - different existing Internet server fault models
  - state-of-art failure detection approaches

- Service replication concepts, approaches and issues
  - different states required to be replicated
  - replication approaches and their major limitations

- Failure recovery approaches and issues
  - failover at Network, Transport, Session and Application level
Conclusion

Questions Raised

- Why, as shown in FT framework constraints figure, the increase of resource does not affect the performance and fault tolerance?

- Why the current FT frameworks lacks transport- nor session/application level failover support despite of the increasing need of next-generation Internet services?

- How content inspection can be used to identify the source of nondeterministic behavior at Application level failover?