REQUEST SUCCESS RATE OF MULTIPATHING I/O WITH A DUAL STORAGE CONTROLLER

Varsha Apte
CSE Department, IIT Bombay

Joint work with

CMG Technical Talk
TCS Gateway Park, Mumbai
October 4th, 2013
Outline

- Introduction
- NetApp HA Pair Architecture
- Client Protocol-SCSI
- Availability Model
- Request Success Model
- Results
- Conclusions
Networked Storage System

- Typical architectural components
  - Networked-clients
  - Storage controllers
  - Disk subsystem
- Availability is an important metric
  - We define it is *probability that an I/O request succeeds*
Outline

- Introduction
- NetApp High Available (HA) Pair Architecture
- Client Protocol-SCSI
- Availability Model
- Request Success Model
- Results
- Conclusions
NetApp HA Pair Architecture

- Two controllers in paired failover configurations
- Each controller owns one LUN
- Failure recovery
  - Failure detection
  - Takeover
    - Disk ownership changes
  - Emulation
  - Giveback
Takeover-Giveback Process

Similar process for non-rebootable failure => involves repair time
Outline

- Introduction
- NetApp HA Pair Architecture
- Client Protocol-SCSI
- Availability Model
- Request Success Model
- Results
- Conclusions
Quantifying Availability

- **What?**
  - Probability that the system is available

- **Why?**
  - Determines weak links in the system
  - To compare different architectures
  - Determines settings of configurable parameters
  - Identifies processes to improve (e.g. reboots, failure detection, etc)

- **Limitation of system availability metric:**
  - Is a metric from system side
Request Success at Client

- Request timeout-retry

- Short periods of unavailability is acceptable (Assume that system will not fail before processing current request)

- Request Failures Per Million (RFPM) under request retry behavior
  - Probability of a request failure
  - Is a client side metric - different from availability

*Motivated by Trivedi et al work on request success availability of SIP server.*
SCSI Protocol with Asymmetric Logical Unit Access (ALUA)

- Client Recovery mechanisms
  - Request-Retry
  - Multipathing

Sends all requests to LUN A

Case: 2: Controller A Rebooting, controller B available

- Request
- Retry
- Multipathing
- Interconnect
- Rebooting
- Controller A

[Diagram showing requests and responses between Client, Controller A, Controller B, LUN A, and LUN B]

SUCCESS
SCSI Protocol with ALUA

Client Recovery mechanisms
- Request-Retry
- Multipathing

Case 3: Both the controllers available and path to controller A failed

Sends all requests to LUN A
SCSI Protocol with ALUA

- **Client Recovery mechanisms**
  - Request-Retry
  - Multipathing

**Case 4: Controller A under repair, controller B emulating controller A**

**Sends all requests to LUN A**

- Port A marked unavailable, try directly on port B
- Timer
- Under Repair Controller A
- Emulating Controller B
- SUCCESS
- Interconnect Request
- Response

Client

LUN A
LUN A

Controller A

Controller B

LUN B
LUN B
This work

- Focuses on calculating *request success probability* while taking into account
  - Controller redundancy and takeover mechanisms
  - Time-out/Retry
  - Multipathing
System Assumptions

- Focus on controller availability
  - Not disk system failures
  - Not interconnect failures
- Two types of failures
  - Rebootable failures
  - Non-rebootable failures
- Coverage probability for takeover, giveback and reboot
- All times are exponentially distributed
- Consider network path failure
- Request processing time is negligible
Overall Approach

- Request succeeds if
  - It finds the LUN available on first try
    - Could be through emulation
  - Finds LUN unavailable on first try
    - But LUN becomes available, on a subsequent retry, before completely timing out
      - Through another path
      - Because its controller reboots
      - Because its controller gets taken over and emulated

These scenarios expressed in the form of a **Probability Tree**

Some values of branch probabilities calculated as **steady state probabilities** of a Markov chain model using **probability of going from one set of states to the other within some time** $t$. 
Outline

- Introduction
- NetApp HA Pair Architecture
- Client Protocol-SCSI
- Availability Model
- Request Success Model
- Results
- Conclusions
Continuous Time Markov Chain

- Captures the state of the system over time
  - As time passes, the system state changes over different states
- Assumes exponential distribution of the time
- At time $t$, the probability of system being in a state is called transient probability $P(t)$.
- Over the time, the probability that the system being in a state converges, i.e. steady state probability.
**Calculation of Availability**

- **Availability**: Steady state probability of either both controllers are up or one is up with emulation

- Available states are:
  - A up, B up
  - A down, B up in emulating mode
  - A up in emulating mode, B down

- Can be easily calculated using tool PRISM

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean time to non-rebootable failure</td>
<td>10000hrs</td>
</tr>
<tr>
<td>Mean time to rebootable failure</td>
<td>500hrs</td>
</tr>
<tr>
<td>Mean time to failure detection</td>
<td>10sec</td>
</tr>
<tr>
<td>Mean time to takeover</td>
<td>30sec</td>
</tr>
<tr>
<td>Mean time to reboot</td>
<td>90sec</td>
</tr>
<tr>
<td>Mean time to repair</td>
<td>3 hrs</td>
</tr>
<tr>
<td>Mean time to auto-giveback</td>
<td>45sec</td>
</tr>
<tr>
<td>Mean time to forced giveback</td>
<td>100min</td>
</tr>
<tr>
<td>Takeover coverage factor</td>
<td>0.9</td>
</tr>
<tr>
<td>Reboot coverage factor</td>
<td>0.9</td>
</tr>
<tr>
<td>Giveback coverage factor</td>
<td>0.9</td>
</tr>
<tr>
<td>Probability of a new request arrives to LUN A or LUN B</td>
<td>0.5</td>
</tr>
<tr>
<td>Probability of port hardware fails</td>
<td>0.00001</td>
</tr>
<tr>
<td>Client retry window</td>
<td>180sec</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- NetApp HA Pair Architecture
- Client Protocol-SCSI
- Availability Model
- Request Success Model
- Results
- Conclusions
Approach: Client Request Modeling

- Use probability tree
  - Nodes represent state of the system that the request finds on arrival, or the state the system will be in after request retries

- Sum of all multiplied probabilities from root to leaf with “success” gives probability of request success
Probability Tree

Assumption: New request finds system in steady state

New request arrives to?

LUN A

(P_{LunA})

LUN B

(1-P_{LunA})

Is port A hardware up?

(2)

Yes

(P_{Aportup})

No

(1-P_{Aportup})

Similar to LUN A

Input to the model

Assume these probabilities 0.5

Steady state probability obtained from CTMC model

Conditional steady state probability given as

\[ P_{Aup6} = \frac{P_{Aportavail \setminus Aup}}{P_{Aportavail}} \]

Assumption: New request finds system in steady state

Input to the model

Assume these probabilities 0.5

Steady state probability obtained from CTMC model

Conditional steady state probability given as

\[ P_{Aup6} = \frac{P_{Aportavail \setminus Aup}}{P_{Aportavail}} \]
Approach: How to Calculate Branch Probability?

- Input to the model

- Steady state probability of system being in favorable(unfavorable) state
  - Can be obtained from CTMC by using PRISM

- Transient probability - The system state changes from unfavorable to favorable within retry window
  - How to get it?
Construct a modified CTMC as follows

Start with original CTMC

Identify states where the system will be, when new request arrives.

Identify states in which system is unfavorable. Initial probability = steady state prob.

Calculate probability of CTMC starting here

Reaching here within retry window time by calculating $P_S(t)$

Remove outgoing arcs from these states.

$S =$ States in which system is up
New request arrives to?

- **LUN A** $(P_{LunA})$
- Similar to LUN A

Is port A H/W up?

- Yes $(P_{Aportup})$
- No $(1-P_{Aportup})$

Is port A available?

- Yes $(P_{Aportavail})$
- No $(1-P_{Aportavail})$

Is controller A up?

- Yes $(P_{Aup6})$
- No $(1-P_{Aup6})$

Response from A?

- Success
- Repair
- Reboot $(1-P_{Areboot8})$

Expands further

- B reboots and either A reboots or B emulates A within $rw$?
  - No $(1-P_{Arecover30})$
  - Yes $(P_{Arecover30})$

Expands further

- Failure
- Success

Initial states $S_{24} = Aportavail \cap Adown \cap Areboot$

Final states, $F_{24} = Aup$

A reboots within $rw$? $(24)$

- Yes $(P_{Arecover24})$
- No $(1-P_{Arecover24})$

Is port B H/W up? $(25)$

- Yes $(P_{Bportup})$
- No $(1-P_{Bportup})$

Is port B available? $(26)$

- Yes $(P_{Bportavail26})$
- No $(1-P_{Bportavail26})$

Is controller B up? $(27)$

- Yes $(P_{Bup27})$
- No $(1-P_{Bup27})$

Response From B? $(28)$

- Reboot $(1-P_{Areboot28})$
- Repair $(1-P_{Areboot28})(1-P_{Arecover28})$

A reboots or B emulates A within $rw$? $(29)$

- Yes $(P_{Arecover29})$
- No $(1-P_{Arecover29})$

Expands further

- Failure
- Success

How to calculate?

**Transient probability different from steady state**

$S_{24} = Aportavail \cap Adown \cap Areboot$

$F_{24} = Aup$

$rw =$ retry window
Calculation of Recovery Probability (CTMC M24)

Initial state probabilities

\[ P_{S_24}^{24}(0) = \begin{cases} \pi_s, & \forall s \in S_{24} \\ 0, & \text{Otherwise} \end{cases} \]

Initial states, \( S_{24} = A_{port avail} \cap A_{down} \cap A_{reboot} \)

Final states, \( F_{24} = A_{up} \)

\( P_{A_{recovery 24}} \) is the probability of CTMC 24 starting in state \( S_{24} \) and reaching \( F_{24} \) within retry window time.
**Probability Tree (cont.)**

**Transient probability different from steady state**

**Initial states,**
\[ S_{24} = A_{\text{portavail}} \cap A_{\text{down}} \cap A_{\text{reboot}} \]

**Final states,**
\[ F_{24} = A_{\text{up}} \]

**Time** \[ 0 \Rightarrow rw \]

**CTMC** \[ rw \]

- remove out going arcs of \( F_{24} \)

**Transient probability of** \( B_{\text{portavail}} \)

at time \( rw \) from steady state \[ ? \]

\[ P_{B_{\text{portavail}}26} = \frac{P_{24}^{24} A_{\text{down}} \cap B_{\text{portavail}}(rw)}{P_{A_{\text{down}}}^{24}(rw)} \]

**Initial states,**
\[ S_{29} = A_{\text{down}} \cap B_{\text{portavail}} \cap B_{\text{up}} \]

**Final states,**
\[ F_{29} = B_{\text{up}} \cap (A_{\text{up}} \cup A_{\text{emulated}}) \]

**Time** \[ rw \Rightarrow 2rw \]

**CTMC** \[ 2rw \]

- remove out going arcs of \( F_{29} \)

**How to calculate?**

\( rw = \text{retry window} \)
Improvement due to dual controllers, and of client protocol

Going from Single Node to HA pair gives huge improvement – 70% reduction in RFPMs (this is also the most costly improvement)

Single node to multipathing and retries, 70% improvement again (this requires very little hardware)

• Can work only for delay-tolerant requests
Sensitivity to reboot time, repair time

- Not very sensitive to reboot time. Why? Our conjecture
  - In case of single node, the repair time dominates the RFPM figures – this sensitivity is seen in the above figure (b)
  - In case of HA pair with retries etc, the takeover mechanism makes rebooting time less relevant
Sensitivity to takeover time, giveback time

• More sensitivity seen to takeover and giveback time
Sensitivity to Detection Delay, coverage factor

- RFPM variation with detection time shows interesting insight –
  - Initial part of the graph shows slight dip – i.e. RFPM decreases with increase in detection delay
  - This is because, when reboot time is low, hasty detection can lead to unnecessary takeover/giveback, which can increase in unavailability
Outline

- Introduction
- NetApp HA Pair Architecture
- Client Protocol-SCSI
- Availability Model
- Request Success Model
- Results
- Conclusions
Conclusions

- RFPM is a client side metric that captures actual outage seen by user
  - Request success probability much higher than estimated without taking client mechanisms into account
- Our methodology gives insights into appropriate parameter settings
  - Administrators should careful in detection delay settings
Future Work

- Scaling the model two n-way controllers
- Distribution of various time durations (takeover, giveback, reboot) should not be assumed to be exponentially distributed
- Effect on performance of I/O requests
  - Response Time
  - Throughput
Thank You