Impact of Software Availability on System Reliability

Carmen Mas-Machuca
Shakthivelu Janardhanan
Yagiz Özkan

September 19-21, 2022, Compiegne, France
Ubiquity and magnitude of software failures

- Software bugs contribute more than 35% of critical network outages [Google2016]
- According to Gartner, the average cost of IT downtime is $5,600 per minute. Amazon may lose millions$ in an hour [Forbes Technology Council, April 2021]

<table>
<thead>
<tr>
<th>Event</th>
<th>Date/Time</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Direct Connect Event in the Tokyo (AP-NORTHEAST-1) Region</td>
<td>02.09.2021 07:30 (JST)</td>
<td><a href="https://aws.amazon.com/de/message/17908/">Link</a></td>
</tr>
<tr>
<td>Twitter outage brings the website offline</td>
<td>14.07.22 7:49AM (CET)</td>
<td><a href="https://bgr.com/tech/twitter">Link</a></td>
</tr>
<tr>
<td>Google Infrastructure Configuration Server operation requests failing</td>
<td>08.05.22 23:45 to 09.05.2022 1:45 (PST)</td>
<td><a href="https://status.cloud.google.com/incidents/2Hd52dn3PqYGTD5zdp7v">Link</a></td>
</tr>
<tr>
<td>Google Infrastructure Configuration Server operation requests failing</td>
<td>10.12.21</td>
<td><a href="https://carleton.ca/polisci/?p=33162">Link</a></td>
</tr>
</tbody>
</table>

The Canadian Centre for Cyber Security (CCCS) issued a security advisory regarding a critical vulnerability → Apache Log4j, a widely used open-source tool for logging and recording activity → would-be attackers to run malicious code on a remote device. → Quebec shut down almost 4,000 websites.
Outline

- Terms and Taxonomy
- Software Dependability Problem
- Addressed questions:
  - How reliable is a new software release?
  - How reliable is a component?
  - How reliable is a system?
- Conclusions
Terms and Taxonomy

Dependability

*Turstworthiness of computing system*

**Threats**

*Factors affecting dependability*

- Fault
- Error
- Failure

**Attributes**

*Metrics to quantify dependability*

- Availability
- Reliability
- Maintainability
- Safety

**Means**

*Ways to improve dependability*

- Fault prevention
- Fault removal
- Fault tolerance
- Fault forecasting

Source: IFIP WG10.4 Dependable Computing and Fault Tolerance https://www.dependability.org/wg10.4/
**Terms and Taxonomy**

- **Fault**: Adjudged or hypothesized cause of an error.
- **Error**: Part of a system state which is liable to lead to failure.
- **Failure**: Deviation of the delivered service according to its specification.

- **Active**: it produces an error
- **Dormant**: it has not produced an error
- **Detected**: it has manifestated as failure
- **Latent**: it has not been detected
Terms and Taxonomy

- **Availability**: The ability of an item to perform its required function, under environmental and operational conditions at a stated instant of time.

- **Reliability**: The ability of an item to perform its required function, under environmental and operational conditions, for a stated period of time.

- **Maintenability**: the probability of performing a successful repair and maintenance action within a given time.

- **Safety**: Ability of an item to provide its required function without the occurrence of catastrophic consequences on the user(s) and the environment.

Source: ISO 8402 and British Standard BS 4778
Terms and Taxonomy

- **Fault prevention** is attained by quality control techniques employed during the design and manufacturing of hardware and software.

- **Fault removal** is performed both during the development phase (verification, diagnosis, and correction), and during the operational life of a system (either corrective or preventive maintenance).

- **Fault tolerance** is intended to preserve the delivery of correct service in the presence of active faults.

- **Fault forecasting** is conducted by performing an evaluation of the system behaviour with respect to fault occurrence or activation: either qualitative (identify, classify, rank the failure modes), or quantitative (probabilities to which some of the attributes are satisfied).

Source: “Fundamental Concepts of Dependability” A. Avizienis et al.
Terms and Taxonomy: Software faults

- Software fault = bug

- Types of software faults:

  - **Bohrbugs** (deterministic)
    - Description: "solid" logical faults
    - Fault handling strategies: Remove
    - Example: *Path Computation Element (PCE) able to create tunnel with negative bandwidth*
Terms and Taxonomy: Software faults

- Software fault = bug

- Types of software faults:

  - **Bohrbugs (deterministic)**
    - **Description**: "solid" logical faults
    - **Fault handling strategies**: Remove
    - **Example**: Path Computation Element (PCE) able to create tunnel with negative bandwidth

  - **Mandelbugs (non-deterministic)**
    - **Description**: "relative" logical faults
    - **Fault handling strategies**: Retry, replicate
    - **Example**: Distributed database locking in ONOS
Terms and Taxonomy: Software faults

- Software fault = bug

- Types of software faults:
  
<table>
<thead>
<tr>
<th>Threats</th>
<th>Faults affecting dependability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bohrbugs (deterministic)</td>
<td>&quot;solid&quot; logical faults</td>
</tr>
<tr>
<td>Mandelbugs (non-deterministic)</td>
<td>&quot;relative&quot; logical faults</td>
</tr>
<tr>
<td>Ageing-related bugs</td>
<td>Degradation with time</td>
</tr>
<tr>
<td>Path Computation Element (PCE) able to create tunnel with negative bandwidth</td>
<td></td>
</tr>
<tr>
<td>Distributed database locking in ONOS</td>
<td></td>
</tr>
<tr>
<td>Flows still reported in oper data store after they have been deleted from both config and network</td>
<td></td>
</tr>
</tbody>
</table>
Limitations of the State of the Art

- **Threat** analysis focus on independent component failures
  - Focused on hardware failures
  - Software related failures *neglected* or *oversimplified* (e.g., as single failure mode)

- **Attributes**, e.g.,
  - *reliability*, does not precisely describe software behaviour
    - *Reliability growth* due to *maturity*
    - *Reliability degradation* due to *ageing*

- **Means** focus on structural protection
  - Fault *prevention, removal* and *forecasting* have been overlooked

Source: IFIP WG10.4 Dependable Computing and Fault Tolerance https://www.dependability.org/wg10.4/
Software Dependability Problem

- Softwarized components/systems/networks
- Open source code
Software Dependability Problem

- Softwarized components/systems/networks
- Open source code

**Target**: Realistic and practical dependability analysis

**Specific problems:**

- How reliable is a new release?
- How reliable is a component?
- How reliable is a system?
Software Dependability Problem

- Softwarized components/systems/networks
- Open source code

**Target:** Realistic and practical dependability analysis

**Specific problems:**

- How reliable is a new release?
- How reliable is a component?
- How reliable is a system?
How reliable is a new release?

Data Collection

Bug trackers/Repositories (e.g., Jira)

Detected bugs
Resolved bugs

- ONOS bugs examples

<table>
<thead>
<tr>
<th>Issue</th>
<th>ONOS-8153</th>
<th>ONOS-6401</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>In progress</td>
<td>Closed</td>
</tr>
<tr>
<td>Priority</td>
<td>Major</td>
<td>Critical</td>
</tr>
<tr>
<td>Affected Versions</td>
<td>2.3.0</td>
<td>1.9.2, 1.10.0, 1.8.5, 1.8.6</td>
</tr>
<tr>
<td>First Affected Version</td>
<td>2.3.0</td>
<td>1.8.5</td>
</tr>
<tr>
<td>Resolution</td>
<td>Unresolved</td>
<td>Fixed</td>
</tr>
<tr>
<td>Create Date</td>
<td>2022-04-08 07:06:16</td>
<td>2017-05-00 09:29:49</td>
</tr>
<tr>
<td>Create Date from Start</td>
<td>2916 days</td>
<td>1114 days</td>
</tr>
<tr>
<td>Resolved Date</td>
<td>None</td>
<td>2017-05-02 21:29:49</td>
</tr>
<tr>
<td>Time to Solve (in hours)</td>
<td>None</td>
<td>313.97</td>
</tr>
<tr>
<td>Time to Solve (in days)</td>
<td>None</td>
<td>13</td>
</tr>
<tr>
<td>Month Number from Project Start</td>
<td>96</td>
<td>37</td>
</tr>
<tr>
<td>Week Number from Project Start</td>
<td>416</td>
<td>159</td>
</tr>
</tbody>
</table>
How reliable is a new release?

Data Collection

- Bug trackers/Repositories (e.g., Jira)
- Detected bugs
- Resolved bugs

Bug History Analysis

- Best Models (e.g., SRGM)
- Expected time between detected/resolved bugs
- Residual bug content

**Residual bug content**

\[ r(t) = E[a - N(t)] = a - m(t) \]

SRGM: Software Reliability Growth Models
How reliable is a new release?

Data Collection

Bug trackers/Repositories (e.g., Jira)

Bug History Analysis

Best Models (e.g., SRGM)

Estimation accuracy improvement

+Code Metrics

Increase estimation accuracy
How reliable is a new release?

**Data Collection**
- Bug trackers/Repositories (e.g., Jira)
- Detected bugs
- Resolved bugs

**Bug History Analysis**
- Best Models (e.g., SRGM)
- Expected time between detected/resolved bugs
- Residual bug content

**Estimation accuracy improvement**
- +Code Metrics
- +Metrics from other codes
- Increase estimation accuracy

![Graphs showing bug detection and resolution over time]
How reliable is a new release?

Data Collection

- Bug trackers/Repositories (e.g., Jira)
- Detected bugs
- Resolved bugs

Bug History Analysis

- Best Models (e.g., SRGM)
- Expected time between detected/resolved bugs
- Residual bug content

Estimation accuracy improvement

- Code Metrics
- Metrics from other codes
- Metrics from prev. rel.

Increase estimation accuracy
How reliable is a new release?

How reliable is a component?

How reliable is a system?
Component reliability considering software dependability

- Hardware & Software
- Software: Proprietary & open-source

Models

Homogeneous Markov Chains

Stochastic Petri Nets/ Stochastic Activity Networks (SANs)
Component reliability considering software dependability

Example: SDN controller

1. Software reliability growth
   - long term variations of software reliability

2. Software ageing
   - short term variations of software reliability

3. Nature of failures
   - manifestation
     - transient failure
     - hang and freeze
     - crash
   - recovery
     - retry - restart - reload

4. Operating system

5. General purpose Hardware
Component reliability considering software dependability

Example: SDN controller → SSA analysis

- At least two controllers are needed to achieve “3-nines” availability

  ![Availability Table]

- Identification of the most critical parameters (local sensitivity analysis)

  ![Critical Parameters Diagram]

  [Ros14] assumed much higher availability of SDN controller
  \[ A > 0.999975 \]

  Critical parameters
  a) External failure rates
     (well studied and documented)
  b) Software ageing rate
     (uncertain, load dependant)
Component reliability considering software dependability

Example: SDN controller → *Failure frequency and downtime distribution*

Around 50 failures per year with total duration of 9.68 hours per year are expected.

- Software failures lead to more frequent, but shorter, outages
  - Software failures account for 84% of all failures, but contribute to only 38% of downtime
  - Hardware failures represent less than 4% of all failures but contribute to 44% of downtime
  - 80% of the failures resulted in outages shorter than 10 min; median being 3.6 min
Component reliability considering software dependability

Example: Switch

- Several components: ASIC, Memory, CPU, Line Cards, Switch fabric, ...
- Each component:
  - Regular HW and SW failures
  - Ageing for HW and SW
Component reliability considering software dependability

Example: Switch

- Proprietary hardware switches (e.g., Cisco)

- Most critical parameters:
  - Memory Ageing and HW_reparation times
  - Other SW Dev ageing and successful repair
Component reliability considering software dependability

Example: Switch

- Software failures lead to more frequent, but shorter outages
  - Software failures account for 80% of all failures, but contribute to only 35% of downtime
  - Hardware failures represent less than 20% of all failures but contribute to 65% of downtime
- Switch availability $\rightarrow 0.9988$  $\rightarrow$ MDT $\sim 10.1$ hours/year
Component reliability considering software dependability

Example: Switch

- Proprietary hardware switches (e.g., Cisco)
- P4 Hardware Target switch (e.g., Intel Tofino)
- P4 Software Target switch (e.g., t4p4s)
Component reliability considering software dependability

Example: Switch

- Proprietary hardware switches (e.g., Cisco)
- P4 Hardware Target switch (e.g., Intel Tofino)
- P4 Software Target switch (e.g., t4p4s)

Switch Availability comparison

Switch Mean Down Time comparison
- P4 software target has higher software failure frequency (92%) than other switches (82%)
- Software failures are faster to repair→ P4 software target switch more MDT due to software failures→ SW_targ is the most critical component
- HW Target switch has faster SW restoration time thanks to their modular SW.
How reliable is a new release?

How reliable is a component?

How reliable is a system?

open source

SW

HW
System reliability

- Aggregation/connected set of components
- First studies towards sovereignty → data center use case
  - Best topology?
  - How many manufacturers?
  - How they should be placed?
System reliability

Data Centers (DC)

DC Topology
- 3 Tier Leaf Spine
- Fat Tree
- AB-Fat tree
- Facebook 4-post
- Facebook Fabric

DC Size
- Small (1K Servers)
- Medium (32K Servers)
- Large (64K Servers)
- Mega (100K Servers)

Arrangement
- Random
- Left-Right
- Left-Right Sequential
- Pod-wise

Traffic

![CDF](image1.png)

![Graph](image2.png)
System reliability

Data Centers (DC)

Different failure scenarios:
- For each layer (ToR, aggregation, core)
- For each manufacturer/set of manufacturers
  - Hardware manufacturers
  - Software developers
    - Native developers
    - Other software developers

Evaluate the impact on the topology connectivity and survivable traffic.
System reliability

Data Centers (DC)

Heat maps and robustness surfaces on connectivity and max-flow between ToR pairs

Compare
- Different topologies
- Different manufacturers
- Different arrangements

Evaluate sensitivity analysis
System reliability

Data Centers (DC)

<table>
<thead>
<tr>
<th>DC Modeling</th>
<th>Failure Generation</th>
<th>Sovereignty Analysis</th>
</tr>
</thead>
</table>

Left-Right Sequential Best

If operator aims at survival traffic

- 210GB → at least 3 manufacturers
- 240GB → at least 4 manufacturers

**Heat Map of Mean Max Flow**

<table>
<thead>
<tr>
<th>Arrangement</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>320 GB</td>
<td>272 GB</td>
<td>238 GB</td>
<td>203 GB</td>
<td>169 GB</td>
<td>136 GB</td>
<td>100 GB</td>
<td>61 GB</td>
<td>34 GB</td>
</tr>
<tr>
<td>Left-Right Sequential</td>
<td>320 GB</td>
<td>281 GB</td>
<td>248 GB</td>
<td>215 GB</td>
<td>187 GB</td>
<td>160 GB</td>
<td>123 GB</td>
<td>85 GB</td>
<td>56 GB</td>
</tr>
<tr>
<td>Left-Right</td>
<td>320 GB</td>
<td>262 GB</td>
<td>207 GB</td>
<td>158 GB</td>
<td>116 GB</td>
<td>82 GB</td>
<td>56 GB</td>
<td>31 GB</td>
<td>14 GB</td>
</tr>
<tr>
<td>Ped-wise</td>
<td>320 GB</td>
<td>262 GB</td>
<td>214 GB</td>
<td>171 GB</td>
<td>130 GB</td>
<td>97 GB</td>
<td>66 GB</td>
<td>43 GB</td>
<td>24 GB</td>
</tr>
</tbody>
</table>
System reliability

Data Centers (DC)

DC Modeling  
Failure Generation  
Sovereignty Analysis

5 HW manufacturers  
5 SW manufacturers

$R_n$: Ratio of man. SW dev. to all SW failures
Data center operators guidelines

In small DCNs (less than 5000 servers) → Leaf-Spine
In larger DCNs → Clos-network-based topology (e.g., fat tree)

The higher the requirements, the more manufacturers are needed → market and law limited

Severity of SW failures → critical parameter to determine number of required developers

The more HW manufacturers, the less *non-native* SW developers are required
Conclusions

Impact of software failures

Ageing and bugs

Presented bottom-up approach
Questions?