Network Availability for Distributed Applications

Luigia Petre, Kaisa Sere, and Marina Waldén
Åbo Akademi University
Department of Information Technology
Network Availability

• Dependability: degree of reliance, justifiably placed on software-intensive systems

• Availability: dependability attribute
  • Readiness for correct service
  • $\text{MTBF}/(\text{MTBF}+\text{MTTR}) \rightarrow$ measuring approach
  • Quality attribute (software architecture term) $\rightarrow$ constructive approach
Availability tactics

Availability

Detection
- Ping/Echo
- Heartbeat
- Exception

Recovery-preparation and repair
- Voting
- Active redundancy
- Passive redundancy
- Spare

Recovery-reintroduction
- Shadow State
- Resynchronization
- Rollback

Prevention
- Removal from service
- Transactions
- Process monitor

Fault → Fault masked
Repair made
Our approach

• Constructive approach to availability
  • Availability constructions correctly integrate onto the application
  • Network availability features for a distributed application
• MIDAS: middleware language
  • Separation of concerns wrt functional/non-functional aspects
  • Functional properties first, then: does availability fit?
  • Abstraction and refinement: central
Modeling framework

- Topological action systems

\[ A = \begin{align*}
&\left(\exp y; \text{var } x; \text{imp } z; \text{do } \left[ \begin{array}{c}
\text{\(i \in I\)} \\
\text{\(a_i\)}
\end{array} \right] \right) \bigg| \\
&N = \text{set of locations, \(\{\lambda\}\) default location}
\end{align*} \]

- Variables: \((v, loc, Val, val)\)

- Actions: \((a, loc, A)\)

\[ A ::= \text{abort} | \text{skip} | v.val := e | b \rightarrow A | A ; A | A \left[ \right] A | \text{if } b \text{ then } A \text{ else } A \text{ fi} | \text{move}(v, \alpha_0, \alpha) | \text{move}(a, \alpha_0, \alpha), \]

where

\[ \text{move}(v, \alpha_0, \alpha) =_{\text{def}} \alpha_0 \in v.loc \rightarrow v.loc := v.loc \setminus \{\alpha_0\} \cup \{\alpha\} \]

\[ \text{move}(a, \alpha_0, \alpha) =_{\text{def}} \alpha_0 \in a.loc \rightarrow a.loc := a.loc \setminus \{\alpha_0\} \cup \{\alpha\} \]
Example

- \( N = \{\alpha, \beta, \eta, \delta, \mu\} \)

\[
A_1 = |[\text{exp } y_1, y_2; \text{ var } x_1, x_2; \text{ imp } z; \text{ do } a_1 [] a_2 \text{ od }]|
\]

\[
a_1@\{\delta\} :: x_1.\text{val}=2 \rightarrow \text{move}(x_1, \eta, \beta) \text{ when } x_1.\text{val}=2
\]
Execution model

• **guard condition** $g(a)$
  • Ensures that $a$ behaves as intended
  • $g(\text{skip}) = \text{true}$, $g(v.val := e) = \text{true}$, $g(b \to A) = b \land g(A)$
  • $a$ **enabled** when $g(a)$ holds

• **Execution**
  • State of $A$ ($x$ and $y$) initialized
  • Enabled actions at $\bigcup_{i \in I} a_i.loc$ non-deterministically chosen and executed, typically updating $x$ and $y$
  • Non-conflicting actions, enabled at the same time: can be seen as executed in parallel
  • Actions are atomic
  • Computation terminates if no action enabled, otherwise continues infinitely
Example

- \( N = \{\alpha, \beta, \eta, \delta, \mu\} \)

\[
A_1 = \left[ \text{exp } y_1, y_2; \text{ var } x_1, x_2; \text{ imp } z; \text{ do } a_1 [] a_2 \text{ od } \right]
\]

\( a_1@\{\delta\} :: x_1.\text{val}=2 \rightarrow \text{move}(x_1, \eta, \beta) \text{ when } x_1.\text{val}=2 \)

- Choose action from \( \{a_1@\{\delta\}, a_2@\{\delta\}, a_2@\{\mu\}\} \)

If \( a_1, a_2 \) both enabled, \( a_2 \) has more chances to be chosen than \( a_1 \)
Some invariants

• Invariant I
  • properties and constraints of $x, y, \bigcup_{i \in I} a_i$, and $x.loc, y.loc, \bigcup_{i \in I} a_i.loc$
  • E.g., used to prove consistency of A: I holds initially and before and after $a_i$ is executed

• Integrity conditions
  • $no_{\text{var}} : \text{Var} \times N \rightarrow \text{Nat}, no_{\text{act}} : \text{Act} \times N \rightarrow \text{Nat}$
    $\forall v \in \text{Var}, \forall \alpha \in N \cdot no_{\text{var}}(v, \alpha) \in \{0,1\}$
    $\forall a \in \text{Act}, \forall \alpha \in N \cdot no_{\text{act}}(a, \alpha) \in \{0,1\}$
Refinement

• Technique for stepwise system development
  • a high-level system model is transformed by a sequence of correctness preserving steps into a more concrete and deterministic model satisfying the original specification

• Algorithmic, data, superposition

• behavior($A$) - set of state sequences that correspond to all the possible executions of $A$.

• Superposition refinement
  • $A_1 \sqsubseteq A_2$, when the behavior of $A_1$ is preserved by $A_2$ and the new behavior introduced by $A_2$ does not influence or take over the behavior of $A_1$. 

Network availability with MIDAS

• Required resources: **accessible** over the network
• Network nodes: **active** (not under maintenance)
• Manage replicated resources
Accessible resources

**Example**: online accessing journal papers

- Granted if the organization (university) has paid a subscription
- If users are travelling outside their network: the online access is only guaranteed if the network they plug into has access to the journal as well
- The same action of printing a paper published by a journal and made accessible online succeeds in the home network but not necessarily outside this network
Accessibility concepts

• **Cell** - set of accessible locations for each $a$ located at a certain location $\{\alpha\} \subseteq N$, typically including $\alpha$
  • $(a,loc,A)$, $A=g\rightarrow B$
  • $\nu A$ - names of all the variables used by $a$
  • $iA$ - names of the used imported variables, $iA \subseteq \nu A$
  • $cell: Act \times N \rightarrow P(N) \cup \{\lambda\}$

• **Access predicate**
  • $access(a@\alpha) =_{def} \forall \nu \in \nu A \cdot (\exists \rho \in cell(a,\alpha) \cdot \nu \in \rho.var \land$ $(\nu \in iA \land \nu.iloc \neq \emptyset \Rightarrow \rho \in \nu.iloc))$
Accessibility example with local variables

- \((a, \{\alpha\}, A), A = (v.val=5 \rightarrow v.val := w.val)\)
- \(cell(a,\alpha) = \{\alpha, \beta, \mu, \tau\}\)
  - \(access(a@\alpha) = (\exists p \in cell(a,\alpha) \cdot v \in p.var) \land (\exists \xi \in cell(a,\alpha) \cdot w \in \xi.var)\)
Accessibility example with imp variables and security issues

• If $v \in iA$ is specified together with its desired locations of import ($v.iloc \neq \emptyset$) => location $\rho$ needs to be one of the desired locations of import $v.iloc$.

• **Example**: we could specify the location of downloading some installation software,
  • *access* predicate checks whether the location of import is indeed the desired one.

• *access* predicate
  • depends on the identity of the action
  • actions $(a1, \{\alpha\}, A)$, $(a2, \{\alpha\}, A)$ may have distinct cells => distinct *access* predicates
  • security restrictions: $a1$ has access to only some locations, while $a2$ has a wider cell.
Cell evolution

- Printing \((a)\) papers \(v\) and \(w\) does not work from \(\eta\): \(cell(a,\eta) = \{\eta\}\)

- VPN + remote desktop connection extend the cell
### Active nodes

- We define predicate $\text{active}(a\@\alpha)$, $\forall (a, \{\alpha\}, A)$, $\{\alpha\} \subseteq N$, $A=g \rightarrow B$
  - $N = N_{act} \cup N_{maint}$, $N_{act} \cap N_{maint} = \emptyset$
  - $\text{locations}(a)$ - set of locations of all the resources involved in $(a, \{\alpha\}, A)$, including $\alpha$, where $\alpha \in a.loc$
    - if $A=move(\nu, \theta, \tau) \Rightarrow \text{locations}(a) = \{\alpha, \theta, \tau\}$
  - models that all the network nodes needed for executing $a$ are active: $\text{active}(a\@\alpha) = \text{locations}(a) \subseteq N_{act}$
Replication example
Replication constructs

• Replicated resource $R$
  • $|R.loc| > 1$
  • replicas of variable $v$, $v.loc = \{\alpha_1, ..., \alpha_n\}$, have the same name, type, and value, but different locations: $v@\alpha_1, ..., v@\alpha_n$

• Actions
  • $\text{copy}(v, \Gamma) = \text{def } v.loc \neq \{\lambda\} \rightarrow v.loc := v.loc \cup \Gamma$
  • $\text{remove}(v, \Gamma) = \text{def } \text{if } v.loc \setminus \Gamma \neq \emptyset \text{ then } v.loc := v.loc \setminus \Gamma$
    \text{else } v.loc := \{\lambda\}$ fi
Enabledness

- **Location guard of** $(a,\{\rho\},A)$:
  \[
  lg(a@\rho) =_\text{def} \ access(a@\rho) \land active(a@\rho)
  \]

- **Guard of** $(a,\{\rho\},A)$:
  \[
  gd(a,\{\rho\},A) =_\text{def} lg(a@\rho) \land g(A)
  \]
  where $g(A)$ - guard condition $(a,\{\rho\},A)$

- $(a,\{\rho\},A)$ is **enabled** if $gd(a,\{\rho\},A)$ holds
Key concept

• We embed the network availability restrictions (resource accessibility, node readiness) onto the distributed application by strengthening the enabledness conditions for the actions.

• General mechanism
  • Other availability concepts as predicates
  • Example:
    • $\text{integrity}(\text{move}(v, \alpha_0, \alpha)) = \text{def} \ no\_\text{var}(v, \alpha) = 0$
    • $\text{lg}(\text{move}(v, \alpha_0, \alpha)@\rho) = \text{def} \ access(\text{move}(v, \alpha_0, \alpha)@\rho) \land \text{active}(\text{move}(v, \alpha_0, \alpha)@\rho) \land \text{integrity}(\text{move}(v, \alpha_0, \alpha))$
Separation of concerns

<table>
<thead>
<tr>
<th>Specification</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>User</td>
</tr>
<tr>
<td>Distributed Application</td>
<td>Application developer</td>
</tr>
<tr>
<td>Network-Aware Application</td>
<td>Network manager</td>
</tr>
<tr>
<td>Implementation</td>
<td>Programmer</td>
</tr>
</tbody>
</table>

- We must ensure that our proposed model for specifying *network availability* is correct
Ensuring correctness

- **Application**\(_\text{Model}\) and **NAV**\(_\text{Model}\)
- Abstraction relation
  \[ \forall v \in \text{Var}, \forall \alpha \in N \cdot \text{no}_\text{var}(v, \alpha) \in \{0,1\} \land \]
  \[ \forall a \in \text{Act}, \forall \alpha \in N \cdot \text{no}_\text{act}(a, \alpha) \in \{0,1\} \land I \]
- Network resource accessibility: \textit{access}()
- Network node status: \textit{active}()
- Replicated resources management: \textit{copy()}, \textit{remove}()

\[ \Rightarrow \text{Application}\_\text{Model} \subseteq \text{NAV}\_\text{Model} \]
Some related work

• Classical notion of availability: (Laprie, 1985; Avizienis, Laprie, Randell & Landwehr, 2004)

• More sophisticated notions
  • service availability features (usage pattern burstiness)
  • providing the service to all the authorized users
  • preventing non-authorized users from accessing a service
  • (Rossebo, Lund, Husa & Refsdal, 2006; International Standards Organization, 2001)
Some comparison

• (Rossebo, Lund, Husa & Refsdal, 2006)
  • Availability - composed of both accessibility and exclusivity: a service should be accessible to all authorized users (only)
  • Availability metric: tuple $X = (X_1, X_2)$
    • $X_1$ measures the exclusivity aspects, $X_2$ the accessibility aspects
    • Exclusivity consists of a conjunction of predicates:
      • “the probability that an authorized user is denied access to the service at time $t$ should be less than $x$”
      • “the number of intrusions at a given time $t$ should be less than $z$”,
Conclusions

- Formal approach to addressing availability issues wrt distributed applications
- Separation of concerns between the functionality of the application and the non-functional property of network availability for the application
  - We specify a distributed application that captures the application requirements, while considering the network generic
  - We ‘plug’ this network-generic application onto a specific network and put forward several issues that need to be ensured in order for the application to run properly
  - We model the network availability issues to ensure by allowing the application to execute less often (only when the supporting network nodes are available) and by adding code that does not take over nor influences the application.
  - By expressing the network availability issues in these forms, one has a correct embedding of network availability onto the distributed application.