Parametric, Probabilistic, Timed Resource Discovery System

2nd SynCoP Workshop

Camille Coti
coti@lipn.fr

LIPN, CNRS UMR 7030, SPC, Université Paris 13

April 3rd, 2016
Roadmap

1. Problem
   - Real-life problem
   - Reservation protocol

2. Modeling the system
   - Modeling each machine
   - Modeling the reservation system
   - Interactions between automata

3. Verification of the system
   - Parameters of the system
   - Expected behavior

4. Conclusion
Real-life problem

Problem: share resources
- For example: computing nodes in a lab
- Exclusive access
- Fair resource sharing

Resource assignment systems
- Reservation systems, batch schedulers: OAR, PBS, Torque, SLURM...
- Use a front-end node: centralized, requires a dedicated node, need to connect to the front-end to access the resources → overhead

In small organizations (one lab, a set of servers...), might not be possible → **Fully distributed** resource discovery and reservation system: QURD.
QURD is *completely distributed*:
- No additional node
- Runs entirely on the computing resources and the clients

**Architecture**

**Users**

**Local network**

**Computing resources**

**ZEROCONF**
Zeroconf

Network protocol

- Originally: self-configuration of network devices
- Extended to other services: DNS, printers...
- Uses UDP multicast datagrams

Operations available:
Zeroconf

Network protocol

- Originally: self-configuration of network devices
- Extended to other services: DNS, printers...
- Uses UDP multicast datagrams

Operations available:

- **Discover**: automatic service detection
  - Client: multicast datagram “who provides this service?”
  - Service: answer (unicast) “me”
Zeroconf

Network protocol
- Originally: self-configuration of network devices
- Extended to other services: DNS, printers...
- Uses UDP multicast datagrams

Operations available:
- **Discover**: automatic service detection
  - Client: multicast datagram “who provides this service?”
  - Service: answer (unicast) “me”
- **Advertise**: also used for service detection, services advertise themselves on the network
  - Service: multicast datagram “here is what I provide”
  - Client: listens to the network and reads the datagrams
Zeroconf

Network protocol
- Originally: self-configuration of network devices
- Extended to other services: DNS, printers...
- Uses UDP multicast datagrams

Operations available:
- **Discover**: automatic service detection
  - Client: multicast datagram “who provides this service?”
  - Service: answer (unicast) “me”
- **Advertise**: also used for service detection, services advertise themselves on the network
  - Service: multicast datagram “here is what I provide”
  - Client: listens to the network and reads the datagrams
- **Resolution**: typically used by mDNS
  - Client: multicast datagram “what is the IP address associated with this name?”
  - Host that has this name: “I am this machine, here is my IP address”
Reservation protocol

Advertise on Zeroconf

- Available machines *publish* themselves on the Zeroconf bus
- When machines are reserved, they *unpublish* themselves

Clients look at **which machines** are available on the network

- Once it has enough machines, it starts its application
- Otherwise: release *(fail semantics)* or wait *(wait semantics)*
Reservation protocol

Advertise on Zeroconf

- Available machines *publish* themselves on the Zeroconf bus
- When machines are reserved, they *unpublish* themselves

Clients look at **which machines** are available on the network

- Once it has enough machines, it starts its application
- Otherwise: release (*fail* semantics) or wait (*wait* semantics)

**Sufficient to have the properties we want?**
Reservation protocol

Advertise on Zeroconf

- Available machines *publish* themselves on the Zeroconf bus
- When machines are reserved, they *unpublish* themselves

Clients look at **which machines** are available on the network

- Once it has enough machines, it starts its application
- Otherwise: release (*fail* semantics) or wait (*wait* semantics)

*Sufficient to have the properties we want?*

*NO!*
Reservation protocol

Advertise on Zeroconf
- Available machines *publish* themselves on the Zeroconf bus
- When machines are reserved, they *unpublish* themselves

Clients look at **which machines** are available on the network
- Once it has enough machines, it starts its application
- Otherwise: release (*fail* semantics) or wait (*wait* semantics)

**Sufficient to have the properties we want?**
**NO!**

State of a machine
- *Available*: can be used
- *Reserved, running, finished*: cannot be used, already taken
1 Problem
   - Real-life problem
   - Reservation protocol

2 Modeling the system
   - Modeling each machine
   - Modeling the reservation system
   - Interactions between automata

3 Verification of the system
   - Parameters of the system
   - Expected behavior

4 Conclusion
Modeling each machine

- Start → Available
- Unavailable

Available

Modeling the system
Verification of the system
Conclusion

Interactions between automata
Modeling each machine

- **unavailable**
- **available**
- **reserved**

- Start → available
- available → reserved
- reserved → available
- available → unavailable
- unavailable → available

- [request?]
- [cancel?]
Modeling each machine

Problem
Modeling the system
Verification of the system
Conclusion

Modeling each machine
Modeling the reservation system
Interactions between automata

Available

Reserved

Running

Unavailable

Start

[request?]

[launch?]

[cancel?]
Modeling each machine

- available
  - request?
    - ack1!
  - cancel?
- reserved
  - launch?
    - ack2!, time:=0
  - [time > T], (1 − λ)
    - ack3!
- running
  - crash, time=τ], (λ)
- dead
- finished

Modeling the reservation system

Interactions between automata
Modeling each machine

Available

Reserved

Running

Finished

Dead

[request?]
ack1!

[cancel?]

[launch?]
ack2!, time:=0

[time > T], (1 - \lambda)
ack3!

[crash, time=\tau], (\lambda)
Resource volatility

Resources can fail while running a process

- Failure probability

\[
\begin{align*}
\text{time} &:= 0 \\
(1 - \lambda) \\
(\lambda) \\
\text{run} \rightarrow \text{sustain} &:= 0 \\
\text{sustain} \rightarrow \text{finish} &:= [\text{time} > T] \\
\text{fragile} \rightarrow \text{dead} &\quad (\lambda) \\
\end{align*}
\]
Modeling the reservation system

begin

\[
\text{cnt} := 0
\]

\[
\text{cnt}++
\]

\[
\text{cb} := 0
\]

\[
\text{timeout}\]

\[
\text{release}\]

\[
\text{cb}++
\]

\[
\text{wait}\]

\[
\text{cb} = \text{NB}
\]

\[
\text{fin} := 0
\]

\[
\text{fin}++
\]

\[
\text{failure}\]

\[
\text{NB}++
\]

\[
\text{done}\]

\[
\text{fin} = \text{NB}
\]

10 / 17 Camille Coti
Parametric, Probabilistic, Timed Resource Discovery System
Modeling the reservation system

\[\text{begin} \quad \text{start} \rightarrow \text{cnt} := 0 \rightarrow \text{reserve}\]

\[\text{cnt}++ \rightarrow [\text{ack}1?]\]

\[\text{cnt}++ \rightarrow \text{launch}\]

\[\text{cnt} = \text{NB} \rightarrow \text{cb} := 0\]

\[\text{timeout} \rightarrow \text{release}\]

\[\text{cb} = \text{NB} \rightarrow \text{wait}\]

\[\text{cb} = \text{NB} \rightarrow \text{fin} := 0\]

\[\text{fin} = \text{NB} \rightarrow \text{done}\]

\[\text{NB}++ \rightarrow \text{failure}\]

Camille Coti
Parametric, Probabilistic, Timed Resource Discovery System
Modeling the reservation system

```
begin
  cnt := 0
  [ack1?]
  cnt++
  [cnt = NB]
  cb := 0
  [timeout]

reserve

launch
```
Modeling the reservation system

start → begin

begin → reserve

cnt := 0

reserve → [ack1?]
cnt++

reserve → [cnt = NB]
cb := 0

launch → [TIMEOUT]

reservation system

Modeling each machine

Modeling the system

Verification of the system

Conclusion

Interactions between automata
Modeling the reservation system

\begin{verbatim}
begin
    cnt := 0
    [ack1?] cnt++
    [cnt = NB] cb := 0
    [ack2?] cb++
release
    [TIMEOUT]

start \rightarrow begin
reserve \rightarrow reserve
launch \rightarrow launch
timeout \rightarrow timeout
\end{verbatim}
Modeling the reservation system

begin

cnt := 0

reserve

cnt := 0

launch

[ack1?]
cnt++

[ack2?]
cb++

wait

[timeout]

timeout

[timeout]

[timeout]

release

fin := 0

fin := 0

cnt := NB

cb := 0

fin := NB

fail := NB++

done

\[\text{cnt} = \text{NB}\]

\[\text{cb} = \text{NB}\]

\[\text{fin} = \text{NB}\]

\[\text{fail} = \text{NB}\]
Modeling the reservation system

```
begin
  cnt := 0
  reserve
    [ack1?] cnt++
  launch
    [cnt = NB] cb := 0
    [ack2?] cb++
  wait
    [cb = NB] fin := 0
    [ack3?] fin++
  timeout
    [TIMEOUT] release

start → begin
```
Modeling the reservation system

begin

start →
cnt := 0

reserve

[ack1?]
cnt++

[ack2?]
cb++

[ack3?]
fin++

[TIMEOUT]

timeout

[cb = NB]
fin := 0

wait

[cb = NB]
fin := 0

failure

[crash]

nb++
Modeling the reservation system

begin

cnt := 0

reserve

cb := 0

[ack1?]

cnt++

[ack2?]

cb++

[ack3?]

fin++

launch

[cnt = NB]

wait

[cb = NB]

[fin = NB]

[TIMEOUT]

NB++

timeout

release

[crash]

done

Conclusion

Problem

Modeling the system

Verification of the system

Interactions between automata

Parametric, Probabilistic, Timed Resource Discovery System
Close-up on the `reserve` state

```
[found]
request!

[done?]
```

Diagram:
- `discover` state
- Transition labeled `[found] request!`
- Transition labeled `[done?]`
Close-up on the reserve state

- [found] request!
- [done?]
- [ack1?]
- cnt++
- cnt=0
- done!
- cnt=NB

Modeling each machine
Modeling the reservation system
Interactions between automata
Interactions between automata

Mini-protocol:

- **Client-resource request**: the client sends a request to each resource it has discovered → request action
Interactions between automata

Mini-protocol:

- **Client-resource request**: the client sends a request to each resource it has discovered → request action
- **Acknowledgement**: each resource contacted answers OK or KO → ack1 action
Mini-protocol:

- **Client-resource request**: the client sends a request to each resource it has discovered → request action
- **Acknowledgement**: each resource contacted answers OK or KO → ack1 action
- **Command**: once the client has all the required resources, it starts the job → launch action
Interactions between automata

Mini-protocol:

- **Client-resource request**: the client sends a request to each resource it has discovered $\rightarrow$ request action
- **Acknowledgement**: each resource contacted answers OK or KO $\rightarrow$ ack1 action
- **Command**: once the client has all the required resources, it starts the job $\rightarrow$ launch action
- **Acknowledgement**: each resource acknowledges it has started its process $\rightarrow$ ack2 action
Interactions between automata

Mini-protocol:

- **Client-resource request**: the client sends a request to each resource it has discovered → request action
- **Acknowledgement**: each resource contacted answers OK or KO → ack1 action
- **Command**: once the client has all the required resources, it starts the job → launch action
- **Acknowledgement**: each resource acknowledges it has started its process → ack2 action
- **Notify done**: once the local process of a resource is done running, the resource notifies the client → ack3 action
Problem
- Real-life problem
- Reservation protocol

Modeling the system
- Modeling each machine
- Modeling the reservation system
- Interactions between automata

Verification of the system
- Parameters of the system
- Expected behavior

Conclusion
Parameters of the system

**Structural parameters**
- Number of (concurrent) clients
- Number of resources

**Application parameters**
- Number of resources used by each job
- Execution time of each process (possibly unbalanced)
- Timeout (*wait* semantics), delay before retry (*fail* semantics)

**Reliability parameter**
- Failure probability
Expected behavior

**Soundness :**
- Option to complete, proper completion and no dead transitions
- Already verified with Petri nets

Specific properties
- **Exclusive access**, no oversubscription
  - Also already verified using Petri nets
Expected behavior

**Soundness**:  
- Option to complete, proper completion and no dead transitions  
- Already verified with Petri nets

Specific properties

- **Exclusive access**, no oversubscription  
  - Also already verified using Petri nets
- **Deadlines**  
  - Time range: “in these conditions, when will all the jobs be done”  
  - Schedulability: “how many jobs can I run on this many machines and still complete within that many time units”
Expected behavior

**Soundness**:  
- Option to complete, proper completion and no dead transitions  
- Already verified with Petri nets

Specific properties

- **Exclusive access**, no oversubscription  
  - Also already verified using Petri nets
- **Deadlines**  
  - Time range: “in these conditions, when will all the jobs be done”  
  - Schedulability: “how many jobs can I run on this many machines and still complete within that many time units”
- **Confidence** in volatile environments  
  - Impossible to be sure the jobs will complete in bounded time  
  - Likelihood to complete before a deadline: “There is a likelihood of 50% that all the applications will be done after $N$ time units, 25% after $2N$ time units, 15% after $3N$ time units and 10% that machines will crash too often for the applications to complete”.
Expected behavior

**Soundness**:
- Option to complete, proper completion and no dead transitions
- Already verified with Petri nets

Specific properties
- **Exclusive access**, no oversubscription
  - Also already verified using Petri nets
- **Deadlines**
  - Time range: “in these conditions, when will all the jobs be done”
  - Schedulability: “how many jobs can I run on this many machines and still complete within that many time units”
- **Confidence** in volatile environments
  - Impossible to be sure the jobs will complete in bounded time
  - Likelihood to complete before a deadline: “There is a likelihood of 50% that all the applications will be done after $N$ time units, 25% after $2N$ time units, 15% after $3N$ time units and 10% that machines will crash too often for the applications to complete”.

**System dimensioning**
- Resource sizing: “How many machines do I need to be able to run that many jobs and be sure they will complete before that many time units”
1 Problem
   - Real-life problem
   - Reservation protocol

2 Modeling the system
   - Modeling each machine
   - Modeling the reservation system
   - Interactions between automata

3 Verification of the system
   - Parameters of the system
   - Expected behavior

4 Conclusion
Conclusion

Use-case: distributed application

- The correctness of the algorithm can be verified

**Parametric**

- Analyze the behavior of the system
- Leave some parameters unknown to dimension the system

**Probabilistic**

- Volatile environment: failure probability
- Also a parameter!

Challenging problem

- Large number of parameters
- Time, probabilities
- Potentially big automaton, made of several supbarts replicated
  → Large state space