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 $\rightarrow$ overhead

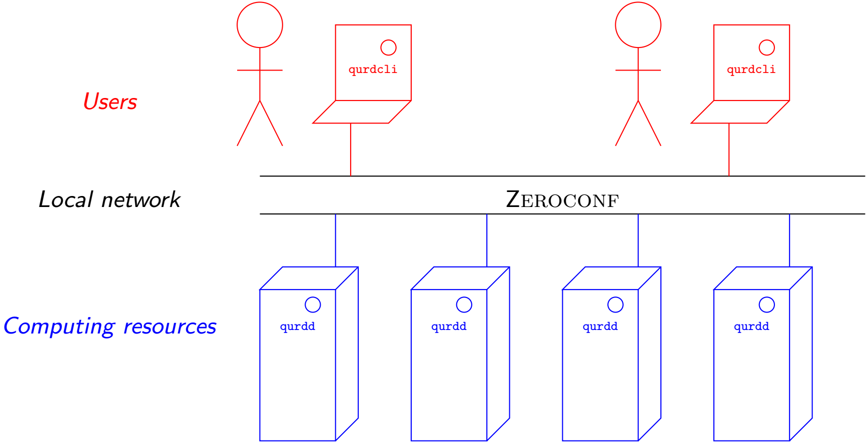
In small organizations (one lab, a set of servers...), might not be possible

$\rightarrow$ **Fully distributed** resource discovery and reservation system: QURD.

Architecture

QURD is *completely distributed*:

- No additional node
- Runs entirely on the computing resources and the clients



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

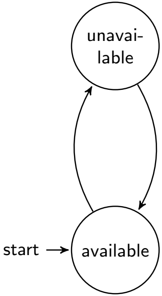
- Once it has enough machines, it starts its application
- Otherwise: release (*fail* semantics) or wait (*wait* semantics)

Sufficient to have the properties we want?
NO!

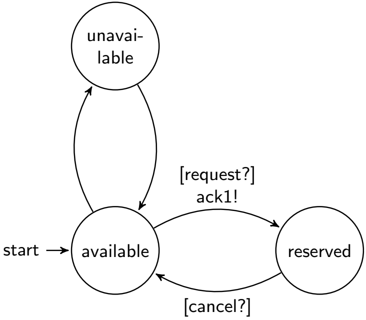
State of a machine

- *Available*: can be used
- *Reserved*, *running*, *finished*: cannot be used, already taken

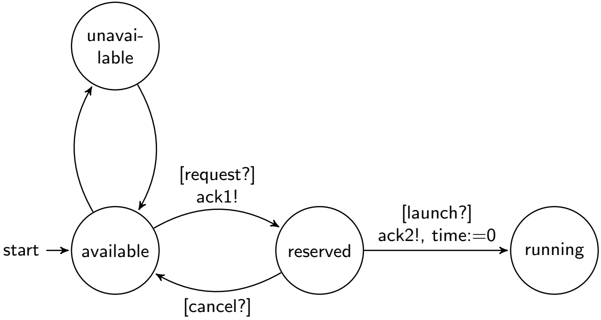
Modeling each machine



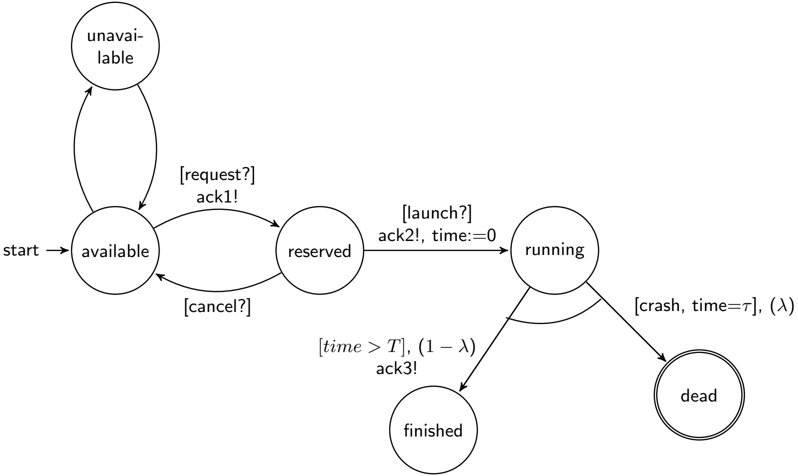
Modeling each machine



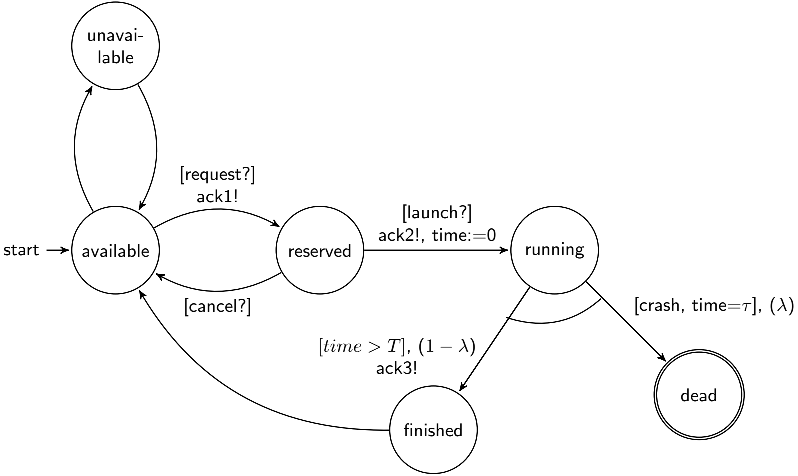
Modeling each machine



Modeling each machine



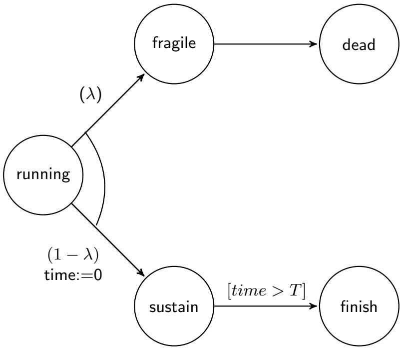
Modeling each machine



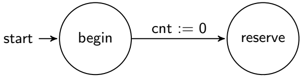
Resource volatility

Resources can fail while running a process

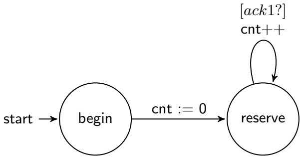
- Failure probability



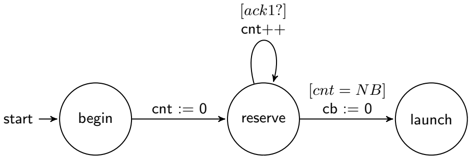
Modeling the reservation system



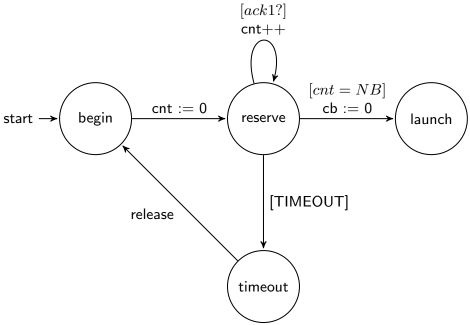
Modeling the reservation system



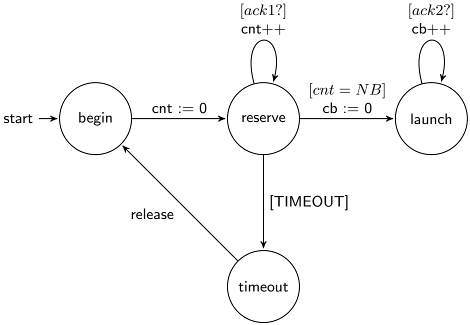
Modeling the reservation system



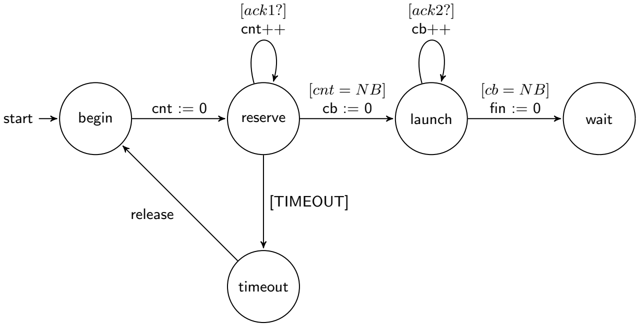
Modeling the reservation system



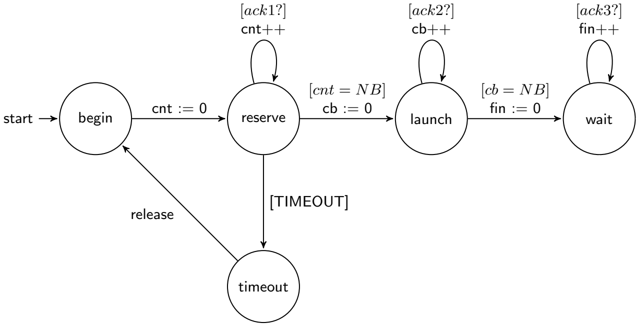
Modeling the reservation system



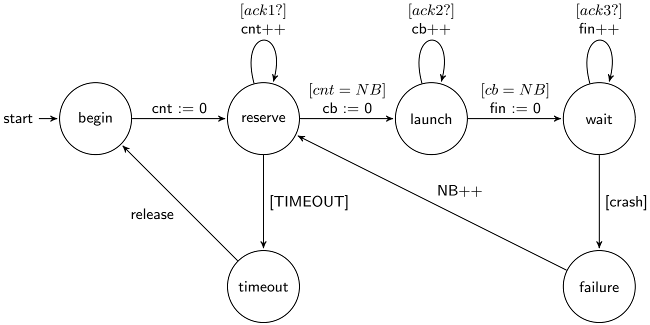
Modeling the reservation system



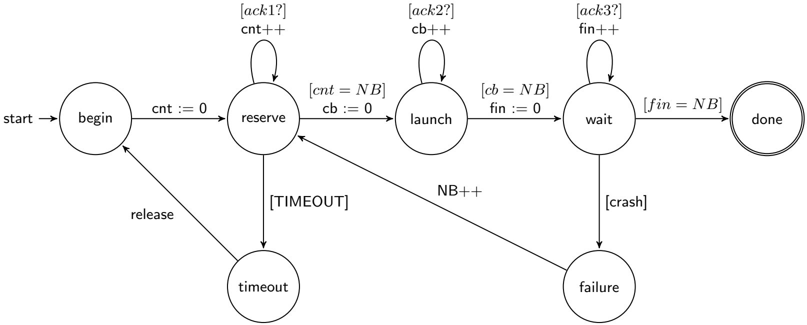
Modeling the reservation system



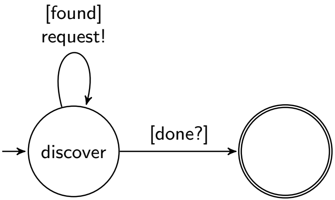
Modeling the reservation system



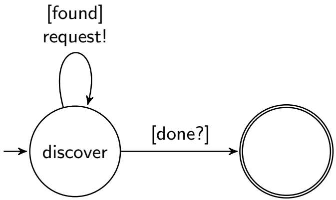
Modeling the reservation system

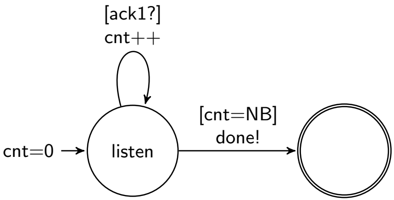


Close-up on the *reserve* state



Close-up on the *reserve* state





Interactions between automata

Mini-protocol:

- **Client-resource request** : the client sends a request to each resource it has discovered $\rightarrow$ `request` 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

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

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

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

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"

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 subparts replicated

→ Large state space