Queuless, Uncentralized Resource Discovery: Formal Specification and Verification

Camille Coti, Sami Evangelista, and Kais Klai

Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030
F-93430, Villetaneuse, France
{*first.last*}*@univ-paris13.fr*

A New Fully Distributed Resource Management System In this paper, we present a formal approach for the specification and the verification of a fully distributed resource reservation system. Our system is made of two parts: the *launcher*, which is executed by the user who wants to run a job on a set of computing nodes, and the *agent*, which is a daemon running on all the resources that exist in the system.

Clients must have an exclusive access to the resources that are allocated for them. Under the requirement that clients have reasonable requirements, all the clients' requests are answered positively in a finite time and all the jobs are executed completely. In order to ensure the correctness of our system regarding such properties, we describe it using a Petri net model, we express formally the desired properties and we perform their formal verification successfully.

Our algorithm relies on the service discovery tools provided by the Zeroconf protocol. Computing nodes declare themselves on the Zeroconf bus. However, this simple discovery service is not sufficient to ensure that the computing resources will not be used by several jobs at the same time.

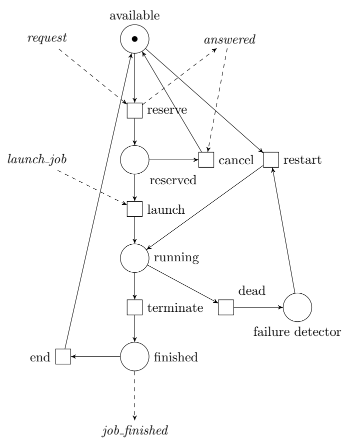

Fig. 1. Model for handling resource volatility with a failure detector

Modelling The Petri net model of a machine is presented on Fig. 1. A machine can be reserved when it is *available*. It answers the client and switches into *reserved* mode. When the local process is done, the machine switches to state *finished*, signals to the client that its part of the job is done, and then returns back to state *available*. There is only one *available* place on each resource, and this place contains only one token in the initial marking. Hence, a machine can answer positively to one client only.

A model where 2 clients issue concurrent resource allocation requests on the same set of resources is represented on Figure 2. Each client has its own reservation system. We represented 2 clients, one requesting n resources and the other m resources.

The *cancel* transition is very important here to release some resources in case of a deadlock caused by a conflict between applications occurring for instance when all the available machines are reserved but no application is able to start. Therefore, after a certain time, if no additional resources appear on the Zeroconf bus, the machines reserved for at least one application will be freed and become available for the other one.

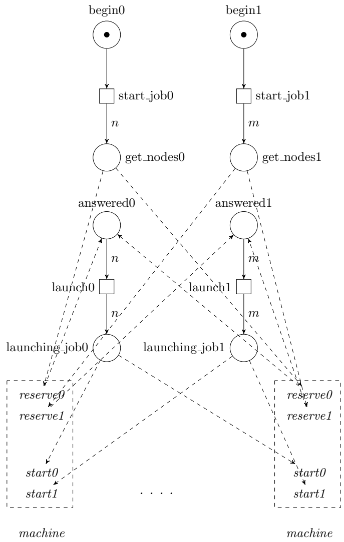

Fig. 2. Reservation system of 2 clients

Analysis We analyzed both generic and specific properties. As generic properties, we were interested in deadlock freeness, boundedness and soundness. The deadlock freeness ensures that no dead state (a state from which no transition is fireable), except the final state (all the jobs are done), is reachable. The boundless property ensures that the number of reachable states is finite. This has been ensured by finding out that the state space of the system has been fully and successfully built in a finite time. Finally the soundness property implies three requirements: (1) *option to complete*, (2) *proper completion*, and (3) *no dead transitions.*

The table right below gives the execution time (in seconds) of Helena and statistical data on their state space : the number of reachable states, the number of terminal reachable states, and the number of arcs in the state space. We selected a set of 6 configurations according to their state space size. A first analysis of the state space report revealed that our model is bounded and that all transitions are executable.

Regarding specific properties, we were interested in checking the following: (1) It is never possible for a machine to be running two different applications, and (2) it is always possible to answer possibly any request (as long as the number of required resources is less than the number of the machines available in the system).

As a conclusion, the properties expected are all verified provided a few reasonable assumptions are made on the environment. First, if we assume that an infinite number of cancellations can not infinitely postpone the beginning of a scheduled job then we can ensure that any submitted job will be scheduled and executed if enough machines are available. Second, in the presence of machine failures, a scheduled job can always terminate if we assume that the pool of available machines allows it.

Configurations				Analysis results			
J	M	P	F	Time	States	Term.	Arcs
4	6	4	no	3.92	1,369,236	1	2,849,412
6	4	2	yes	5.90	2,865,804	1,999	5,740,698
5	6	4	no	13.10	8,407,677	1	17,557,805
4	6	4	yes	20.08	12,111,398	559	27,376,192
6	6	4	no	85.25	43,094,470	1	90,124,518
5	6	4	yes	164.36	65,633,194	1,743	151,096,440