Performability Evaluation of Multipurpose Multiprocessor Systems: The "Separation of Concerns" Approach

Mourad Rabah, Karama Kanoun

Accepted for publication in IEEE Transactions on Computers
Special Issue on Reliable Distributed Systems

Performability Evaluation of Multipurpose Multiprocessor Systems: The "Separation of Concerns" Approach

Mourad Rabah and Karama Kanoun

LAAS-CNRS — 7, Avenue du Colonel Roche — 31077 Toulouse Cedex 4 — France
Tel: +33/561 33 6235, Fax: +33/561 33 6411
kanoun@laas.fr

Abstract

The aim of our work is to provide a modeling framework for evaluating performability measures of Multipurpose, Multiprocessor Systems (MMSs). The originality of our approach is in the explicit separation between the architectural and environmental concerns of a system. The overall dependability model, based on Stochastic Reward Nets, is composed of i) an architectural model describing the behavior of system hardware and software components, ii) a service-level model and iii) a maintenance policy model. The two latter models are related to the system utilization environment. The results can be used for supporting the manufacturer design choices as well as the potential end-user configuration selection. We illustrate the approach on a particular family of MMSs under investigation by a system manufacturer for Internet and e-commerce applications. As the systems are scalable, we consider two architectures: a reference one composed of sixteen processors and an extended one with twenty processors. Then, we use the obtained results to evaluate the performability of a clustered system composed of four reference systems. We evaluate comprehensive measures defined with respect to the end-user service requirements and specific measures in relation to the Distributed Shared Memory paradigm.

Index terms. Dependability and Performability Evaluation, Stochastic Reward Nets, Modular Modeling, Multipurpose Multiprocessors Systems, Distributed Shared Memory, Clustered Systems.

1. Introduction

Computer systems are becoming more and more complex. Even though particular applications may need the design of proprietary systems to achieve the required functions, economic reasons promote the use of support systems available on the market. This situation gave rise to a category of systems that are referred to as multipurpose systems in this paper. Such systems are — to some extent — application-independent support systems. Many system providers have developed generic support systems that can be used in several application domains, for example for instrumentation and control or Web related applications.

Actually, most of the time such systems are composed of Commercial Off-The-Shelf components (COTS). However, despite their wide use, COTS components are far from being highly dependable. As a consequence, a careful design, development and validation of COTS-based systems are needed to provide dependability and high-performance. Indeed, the end-users of a given system want at the same time high dependability and performance, ease of use and low cost. Furthermore, low dependability or low performance of a system will ruin the manufacturer's reputation. Hence the importance of the evaluation of combined dependability and performance measures referred to as performability measures [20].

Our work was motivated by the desire of a system manufacturer to evaluate the performability of a family of Multipurpose Multiprocessor Systems (MMSs) under development, intended for Internet and e-commerce applications. The considered MMS family is scalable, most of its components are COTS and it features a distributed shared memory. A reference architecture composed of 16 processors grouped into 4 nodes has been defined. It can be used for various applications requiring different performance and/or dependability levels. Based on this reference architecture, a whole family can be designed for applications necessitating either higher performance or higher dependability or both. In particular, up to 8 reference architectures can be grouped to form a clustered system. Our first aim was to provide a framework for modeling the

performability of different systems of this family. However, the approach presented in this paper is more general and can be used beyond this particular family of systems.

Even though an MMS is application-independent, usually the system manufacturer targets some particular classes of utilization. He expects to provide high dependability and performance, at least for these targeted utilizations. It is thus important to take them into account for evaluating system performability.

The originality of our approach is in the explicit separation between the architectural and the environmental concerns of a system. In this way, we can clearly analyze the impact of the architectural choices on system performability while the end-user context is taken into account explicitly. Moreover, the results can be used for supporting the manufacturer design choices as well as the potential end-user configuration selection.

This approach allows us to consider two sets of performability measures:

- *Comprehensive measures*, defined with respect to service accomplishment levels that are directly related to the end-user expectations.
- *Specific measures*, related to specific features of the system, interesting particularly the system manufacturer for tuning the support system architecture, to study for example the impact of using a distributed shared memory.

This modeling approach is presented briefly in this paper and applied to two systems pertaining to the MMS family defined by the system manufacturer who initiated this work. The results presented illustrate the kind of outcomes that can be obtained from the MMS performability evaluation. They show the impact of architectural solutions as well as the impact of different utilization environments on system performability. This paper elaborates on our previous work reported in [25, 26] ([25] was dedicated to the specific measures related to the distributed shared memory, while [26] has not addressed the clustered system and the specific measures).

In the following, Section 2 describes our modeling approach and presents current approaches to model construction. Section 3 introduces the systems under consideration and Section 4 defines

their performability measures. Section 5 illustrates the modeling approach through an example and Section 6 presents some results. Finally, Section 7 concludes the paper.

2. Modeling Approach

The model built for evaluating system dependability and performance is always a tradeoff between faithfulness (i.e., correct representation of the real systems behavior) and tractability (i.e., ability to solve the model equations to obtain the measures) [19]. If the system to analyze is too large, a detailed model representing its in depth behavior may be faithful but might be non-tractable because of its size. The use of an appropriate modeling technique such as the Generalized Stochastic Petri Nets (GSPNs) [1] offers a good support. However, it is necessary to master the complexity of the resulting state space by the way of structured modeling approaches. This is one of the aims of this paper in which modeling is based on Stochastic Reward Nets (SRNs) [13], that are obtained from GSPNs by assigning reward rates to tangible markings.

In the rest of this section, we present our modeling approach and outline some other modeling approaches.

2.1 Separation of Concerns Approach

Even though it is possible to model a multipurpose system without considering the end-user requirement in terms of expected services, the results are more accurate if the environmental aspects are incorporated. Hence, our modeling approach takes into account explicitly the environmental context together with the architectural concerns. This is done in a separate manner to allow for some flexibility.

The overall dependability model is composed of two parts, one of them describes the architectural behavior of system components and the other one expresses the system utilization environment (reflecting the end-user point of view). The latter incorporates several kinds of information. A

model is dedicated to each kind of useful information. For our purpose, we consider two models corresponding to the service-level and maintenance policy models.

The global model is thus seen as a composition of three models as depicted in Figure 1. These models are more detailed hereafter. The interactions between the various models is illustrated through a small example in Section 5.

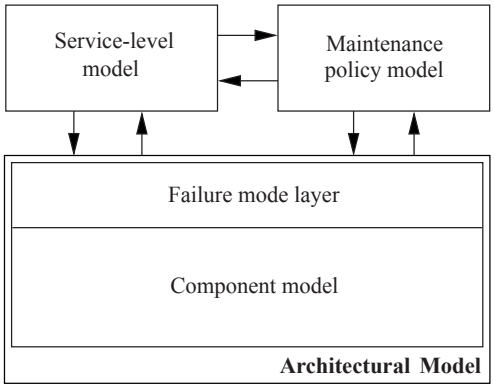


Fig. 1. Model overview

2.1.1 Architectural Model

The architectural model describes the behavior of system components (including all basic hardware and software components as well as fault tolerance mechanisms) as resulting from elementary failures, repairs and interactions regardless of system utilization environment.

From a practical point of view, the failure modes of the system are defined as a function of the states of the system components. Each failure mode is represented by a particular place in the architectural model. The set of failure mode places is grouped into a separate layer, referred to as the *failure mode layer*. The architectural model is thus composed of the *component model* and the *failure mode layer*.

Component model

The component model may be built using any existing modeling method based on GSPNs, such as the ones we have defined in our previous work (see e. g., [6, 16, 17]). The only requirement for our approach is to make all failure mode states available at the failure mode layer, to facilitate interfacing with the service-level and maintenance policy models. In [24], we have defined rules for

constructing efficiently the component model. In particular, we recommend building a sub-model for each component or group of related components (depending on the desired level of detail) describing their behavior and evolution. Sub-model interfacing rules assisting model construction have been defined.

Failure Mode Layer

In addition to the places representing the architectural failure modes, the failure mode layer gathers places summarizing some component states for specific measure evaluation (such as the states of the distributed shared memory). Initially, all places in the failure mode layer are not marked. The failure mode layer is used to facilitate the interface between the architectural and environmental models. All the information considered necessary for the environmental models should be made available in the failure layer model. Also, the information required for component model from the service-level and maintenance policy models is available in the failure mode layer. However, in order not to duplicate unnecessarily some places we have decided to accept that the firing of some transitions, in the component models, could be enabled directly by the marking of very specific places (that are clearly defined) in the service-level and maintenance models, this is illustrated in Section 5. Indeed, in [24], we have defined a set of consistent rules managing the links between the various models.

2.1.2 Service-Level Model

The system service levels are defined with respect to the different service accomplishment and degradation state classes accepted by the considered end-user application. They reflect the application requirements in terms of resources and are usually established according to the architecture failure modes and to the application nature. The service levels are thus partitions of the architecture failure modes.

In the service level model, places represent the various service levels as seen by the end-user. One and only one place can be marked at any time (including the initial marking) indicating the system

current service level. Transitions between the various service levels are entirely governed by the architectural model through the failure mode layer and by the maintenance policy model.

Indeed, the service-level model is not autonomous. Its main role is to provide a high-level vision of the system states according to the end-user expected services. It gives the logical behavior of the system as seen by the end-users.

2.1.3 Maintenance Policy Model

The maintenance policy model describes the maintenance types (e.g., immediate, delayed or programmed), repair resources availability and might include (re)booting procedures, according to the service level and/or the architectural failure mode. The interconnection between the service-level and maintenance policy models represent for example calls for maintenance or notification of maintenance end.

2.1.4 Benefits of our Modeling Approach

Even though modeling is performed from a system manufacturer point of view, we incorporate explicitly the end-user point of view in order to evaluate comprehensive performability measures that are especially relevant to the end-user, in addition to specific measures that may be too detailed for the end-user but of prime interest for the system manufacturer.

Indeed, the same model could be used to evaluate comprehensive and specific measures. Comprehensive measures are evaluated based on the service-level model whereas specific measures are evaluated based on the failure mode layer.

The *separation of concerns* property allows reuse of the architectural model, when the same architecture is to be used in different environments. Obviously, changing the architectural model will lead to change its links with the service-level and maintenance models. However, as these links are clearly identified and formalized, the changes can be performed more easily than when the various models are not separated.

From the end-user perspective, this approach allows the reuse of the environmental models to compare different architectures for choosing the most suitable one for his/her own environment.

Moreover, building the component model in a modular way favors reuse of some component sub-models in the component model, which is of prime interest for both system manufacturers and the end-users.

2.2 Other Modeling Approaches

Our work is related to performability modeling of multipurpose multiprocessors systems. Several publications have been devoted to dependability or performance modeling of multiprocessor systems and more generally of complex computer systems, based on GSPNs and their off-springs.

Considering multiprocessor systems, despite the large number of published work (see e. g., [1] and [22]), the manufacturer and the end-user concerns have not been explicitly addressed at the same time (i.e., the multipurpose paradigm was not studied in depth as in our work).

Separation of abstraction views allows the same architecture to be studied for different service needs and, similarly, several architectures to be compared for the same environmental conditions, changing only a part of the initial model (and its links with the other parts). To the best of our knowledge, the hierarchical composition method [9] is the only method that considers the system under several points of view.

More generally, concerning computer system dependability and performance modeling, the main difficulty for model construction results from the large number of states of the associated model. Several techniques have been published to overcome it. They can be grouped into two categories: *largeness tolerance* and *largeness avoidance* techniques [32].

The main objective of largeness tolerance techniques is to master the generation of the global system model through the use of concise specification methods and automated generation of models. The basic idea is to generate the model of a modular system by composition of the sub-models of its components. Several other authors have published such techniques (e.g., see [21, 27]). Also, numerous evaluation tools using GSPNs and their off-springs have been developed (e.g., SPNP [12], SURF-2 [5], SHARPE [28], UltraSAN [29] and DEEM [7]). GSPNs and their off-

springs have been used to model real-life systems such as air traffic control systems [15, 18], space applications [8] and RAID storage systems [30].

Largeness avoidance techniques try to circumvent the generation of very large models. The basic idea is to construct small sub-models that can be processed in isolation. The results of the sub-models are integrated into a single overall model that is small enough to be processed. Among them, we have for example: the behavioral decomposition technique [4], the hybrid hierarchical modeling technique [3], the data structure technique for the Kronecker solution of GSPNs [11], the method of Superposed GSPNs [14] or of asynchronously communicating modules [10]. To the best of our knowledge, most of these techniques are efficient when the sub-models are loosely coupled and become hard to implement when interactions are too complex.

From a practical point of view, in our modeling framework, as in [9, 23] we combine both techniques for modeling the considered MMS. We use a largeness tolerance technique for building the model of the non-clustered systems (as the various components have several inter dependencies) and a largeness avoidance technique for the clustered multiprocessor systems.

3. Systems under Consideration

We first present the reference and extended architectures of the considered Multipurpose Multiprocessor System (MMS) family with their failure modes, hypothetical end-users, as well as associated service levels and maintenance policies. Then the clustered architecture is defined.

3.1 Reference and Extended Systems

The reference architecture is composed of 16 processors grouped into 4 identical nodes. The extended architecture features a fifth node used as a spare. Since the four basic nodes of the extended architecture are used in the same manner as in the reference architecture and the spare node replaces the first failed node, the remainder of the section will focus on the reference architecture.

3.1.1 Reference Architecture

As depicted in Figure 2, the nodes are connected through an interconnection network composed of two redundant rings. Each node is connected to each ring via a controller. To ensure high availability, all nodes are connected to a centralized redundant diagnostic equipment (DEq). Its role is to log and analyze all error events, initiate and control system reboots. It does not contribute to service delivery but the system cannot reboot without it.

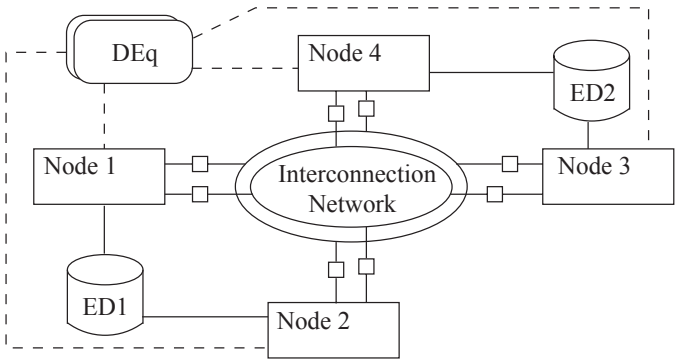


Fig. 2. MMS reference architecture

The MMS has two external RAIDs (Redundant Array of Independent Disks), ED1 and ED2, with on-line maintenance. Each RAID is shared by two nodes. Nevertheless, if a node or its interconnection controller fails, the RAID can be used by the other node.

Figure 3 shows that each node[1] has: i) four independent processors (with a two-level private cache where data from other node memories can be replicated or migrated), ii) an internal disk (ID) devoted either to the operating system (nodes 1 and 3) or to the swap (nodes 2 and 4), iii) a set of dedicated controllers (for interruption, reboot, etc.), iv) a local Random Access Memory, RAM (composed of four banks), v) a Remote Cache Controller (RCC) interconnecting the node to the network and vi) a set of miscellaneous devices, such as local and Peripheral Component Interconnect (PCI) buses, etc.

1 Even though the model of the whole architecture is not presented in this paper, the details of the architectures are briefly outlined to give an idea about the level of details taken into account in the models.

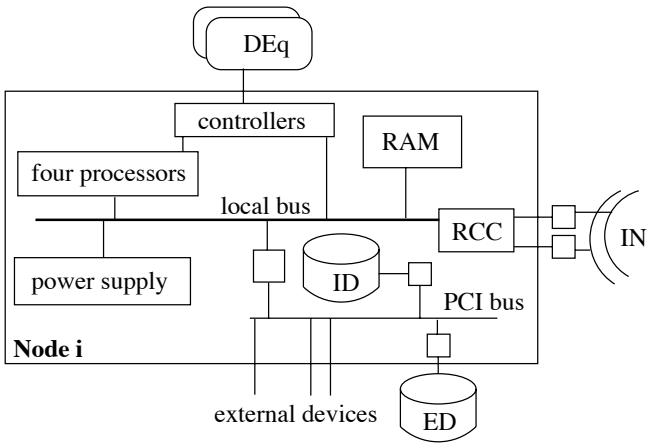


Fig. 3. Main components of a node

3.1.2 Fault tolerance facilities

In addition to the redundant diagnostic equipment, DEq, and to the redundancy of several components (interconnection network, power supply, internal and external disks), MMS includes other facilities to reduce system downtime following error detection, among which:

- Architecture automatic reconfiguration: the load of a failed processor is partitioned among the other processors and the load of a whole node can be partitioned among the other nodes (and the failed node is isolated).
- Reboot with all on-line tests (complete reboot) or with a reduced test set (fast reboot).

It is intended that some failures (whose nature is to be determined by the system manufacturer according to some criteria) will lead to system automatic reconfiguration without or with fast reboot. However, we assume in this paper that any component failure necessitating system reconfiguration (for instance, node, processor, memory bank, some of the controllers failures) forces a *complete reboot*. This assumption can be easily modified for those failures needing system reconfiguration without or with a fast reboot.

A node is available as long as at least one processor, one memory bank, one power supply unit, the local and PCI buses with their controllers are available, otherwise, the node is considered as lost and is isolated. The components of the isolated node become unavailable to the rest of the system.

For modeling purpose, we have associated a coverage factor to each fault tolerance mechanism. The coverage factor is defined as the probability of system recovery given the fact that a fault has been activated or has occurred.

3.1.3 Architectural Failure Modes

Given the large number of components and the various impact of their failure on system state and performance, a multitude of failure modes can be defined, related to the nature of the failed components. We have carried out a Preliminary Risk Analysis to investigate the consequences of all component failures, individually and in combination. The seven identified failure modes are given in Table 1 among which D0, DD, D1, D2 and D3 correspond to performance degradation states and C to system total failure. B gathers the set of states in which one or more components among the redundant components are lost, but the processing capacity of the architecture is not impaired. B is referred to as the benign failure mode. Two additional states are added for modeling purpose: the nominal state (OK) and the reboot state (Reb).

The defined architectural failure modes apply to both reference and extended architectures.

Table 1. Failure modes of the reference system

B	benign failure mode
D0	loss of processors, memory banks or connections to external devices without the loss of an entire node or loss of DEq
DD	loss of the two external disks
D1, D2, D3	loss of 1, 2 or 3 nodes, respectively
C	loss of the four nodes or of a critical component such as the interconnection network

3.1.4 Distributed Shared Memory

The system features a distributed shared memory, composed of the four RAMs of the nodes (i.e., the sixteen memory banks of the system). Each bank is accessible to all nodes. However, as the overall memory is directory based using cache coherent non-uniform memory access (CC-NUMA), the access time to a remote bank is longer reducing system performance. A node can thus be maintained available as long as it has at least one memory bank available among the 4 local banks

(when the four memory banks are lost, the node is considered as unavailable). A memory bank is lost either due to the failure of the memory bank itself or due to the failure of other system components making this bank non-accessible.

When several memory banks are lost, the overall workload is reorganized. The system takes into account the unavailable banks and attributes low memory consuming tasks to the nodes with several unavailable memory banks. Thus, we assume that the state with one failed memory bank in three nodes is equivalent to the state where three memory banks failed in one node.

To evaluate system performability from the distributed shared memory point of view, we have defined sixteen *specific failure modes*, corresponding respectively to the states where i memory banks are lost (denoted Mem_i (i = {1, 2,...,16}). Nevertheless, the refinement of specific failure modes is possible to take into account particular unavailable memory bank combinations.

These specific failure modes are identified explicitly to allow the evaluation of the impact of memory failure on system performability.

3.1.5 End-Users and Service Levels

The above MMS properties and the failure modes are user-independent. However, even though graceful degradation of the architecture is provided by the supporting architecture, this possibility can be exploited differently by the end-users for different application requirements. For instance, some applications will accept a degraded service provided by three, two or even only one node, while others will only accept the loss of some processors, without loosing a whole node. Indeed, the service levels are derived from the architectural failure modes and take into account the end-user application needs.

It is worth noting that failure modes B and C have the same meaning for all applications: in B the entire service is delivered (*ES*) while in C there is no service (*NS*). Between B and C, several service degradation levels can be defined according to end-user needs and to the architecture failure modes.

To illustrate the mapping between architectural failure modes and service levels, we consider two examples of users, denoted X and Y (for which the mapping is given in Table 2).

Table 2. Failure modes and service levels (reference system)

Service levels	Entire Service (ES)	minor Degrad. (mD)	Major Degrad. (MD)	No Service (NS)	Reboot
user X	OK, B	D0, DD	D1	D2, D3, C	Reb
user Y	OK, B, DD	—	—	D0, D1, D2, D3, C	Reb

- **User X**: The service is considered as entire in states OK and B. X accepts a degraded service as long as at least three nodes are available without a critical component failure: NS corresponds to D2, D3 and C. Between ES and NS, we have defined two significant service degradation levels: minor and major degradation levels (respectively referred to as mD and MD). X could be used as a Web server. In ES the full server capacity is available, in mD some transactions are not serviced due to service capacity reduction, in MD less transactions can be processed and in NS, we assume that the Web server should be repaired entirely.
- **User Y** needs the entire capacity of the system and does not accept any service degradation. The entire service is delivered in OK, B and DD. NS gathers all other failure modes.

The states belonging to the same service degradation level are grouped into a state class. These classes are denoted respectively: S_{ES}, S_{mD}, S_{MD} and S_{NS}.

3.1.6 Maintenance

Although maintenance management is user-dependent, the system supplier usually proposes a maintenance contract. In order not to suspend system activities after each failure and call for a paying maintenance, two maintenance policies are defined:

- *Immediate maintenance*: for the states where no service is available, state class S_{NS}. However, as some time is needed for maintenance team arrival on site. Immediate maintenance is performed in two steps: *maintenance arrival* and *system repair* .

- *Delayed maintenance*: for the other states with service degradation. The system continues to be used until a time interval in which its activities can be stopped for repair. Different delays may be allocated to different service levels. Delayed maintenance is also performed in two steps: a maintenance delay during which a degraded service is delivered and repair itself.

The repair is followed by a system reboot.

3.2 Clustered Systems

The manufacturer plans to use the reference architecture as a building block in more complex clustered systems. Architectures composed of 64 (and up to 128) processors distributed among 4 clusters of 4 nodes (respectively, up to 8 nodes) were anticipated from the design stage. The clustered system depicted in Figure 4 is composed of 4 clusters where each cluster is an instance of the reference architecture.

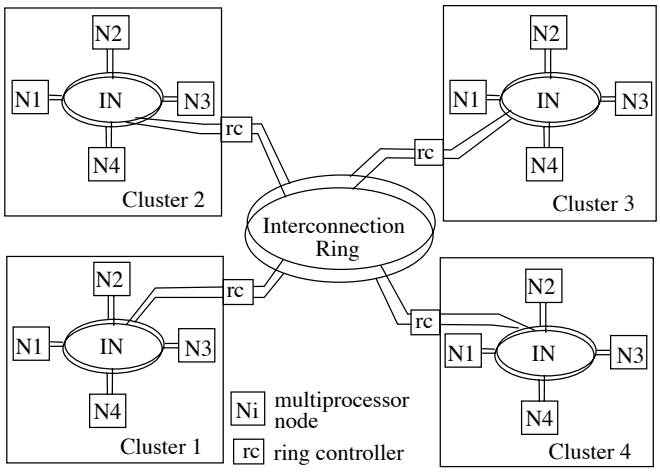


Fig. 4. Clustered architecture

The clusters have similar failure modes as the ones defined for the reference architecture. The clustered system failure modes are defined in Table 3. Also, we defined OK*c* state where all clusters are in their OK state and Reb*c* state the simultaneous reboot of all the clusters.

3.3 Summary of the Considered Systems

In our study we have considered 5 systems: i) two are composed of 4 nodes each, with two different users, ii) two others are composed of 5 nodes each (the fifth one being a spare node) with the two same users X and Y and iii) the clustered architecture, composed of 4 reference systems. They are recalled in Table 4.

Table 3. Failure modes of the clustered architecture

B*c*	at least one cluster is in failure mode B, the others are in the OK state or in B.
D0*c*	loss of some components without loosing an entire cluster
D1*c*, D2*c*, D3*c*	loss of k clusters, k = {1, 2, 3}
C*c*	loss of the four clusters or of the interconnection ring

Table 4. Considered systems

System	Definition
X*r*	reference architecture (4 nodes) for user X
Y*r*	reference architecture (4 nodes) for user Y
X*e*	extended architecture (5 nodes) for user X
Y*e*	extended architecture (5 nodes) for user Y
	Clustered system

4. Dependability and Performability Measures

As stated earlier, we consider comprehensive and specific measures. Comprehensive measures are defined according to service levels. Specific measures concern specific features of the system. In this study, they concern mainly the distributed shared memory.

4.1 Comprehensive Measures

We define for X and Y four common dependability measures:

- two conventional measures noted: *A* for availability and *UA* for unavailability;
- two measures refining the unavailability: UA_{NS} for the unavailability due to one of the S_{NS} class state failure modes and UA_{Reb} for the unavailability due to the reboot procedure.

Furthermore, the availability measure for X is refined into three *dependability levels* corresponding to the three service levels (ES, mD and MD). Table 5 presents the defined measures for X and Y according to the partitions defined in Section 3.1.5 (where *e(t)* denotes the system state at time *t*).

Table 5. Measures for the reference architecture

a- Measure definition

Measure	Definition
LE(t)	dependability level associated with the entire service, ES, at time t
Lm(t)	dependability level associated with the minor service degradation level
LM(t)	dependability level associated with the major service degradation level
A(t)	overall availability
$UA_{NS}(t)$	unavailability due to the S_{NS} class
$UA_{Reb}(t)$	unavailability due to the reboot
UA(t)	overall unavailability

b- Measure equation in terms of Probability (Prob)

Measure	For user class X	For user class Y
LE(t)	Prob $\{e(t) \in S_{ES}\}$	Prob $\{e(t) \in S_{ES}\}$
Lm(t)	Prob $\{e(t) \in S_{mD}\}$	–
LM(t)	Prob $\{e(t) \in S_{MD}\}$	–
A(t)	$LE(t) + Lm(t) + LM(t)$	$LE(t)$
$UA_{NS}(t)$	Prob $\{e(t) \in S_{NS}\}$	
$UA_{Reb}(t)$	Prob $\{e(t) \in S_{Reb}\}$	
UA(t)	$UA_{NS}(t) + UA_{Reb}(t) = 1 - A(t)$	

The steady state measures are respectively denoted: LE, Lm, LM, A, UA_{NS}, UA_{Reb} and UA.

These measures are usually expressed in terms of the probability that the system is considered in service-level state class. However, they can also be transformed into the cumulative time per year spent in the corresponding service-level state class.

Since the system continues to operate in the presence of component failures, with performance degradation, *performability* measures are of prime interest as emphasized in [2], [21] and [31] for practical cases. We assume here that all states in a given service level have the same reward rate.

Let r_X denote the reward rate of the service level X (corresponding to the state class S_X): r_X is the performance index in the class state S_X. The expected reward rate, *E[W]* is defined by:

$$E[W] = r_{ES}\, LE + r_{mD}\, Lm + r_{MD}\, LM + r_{UA}\, UA \qquad (1)$$

The various indexes can be estimated by the end-user, according to the current application. They can, for example, be equal to the ratio of the maximum number of transactions that can be processed by the system in the given service level with respect to the maximum number of transactions in ES.

In the rest of the paper, we will evaluate the steady state unavailability, *UA* (composed of the steady state unavailability due to no service class state, UA_{NS}, and the steady state unavailability due to system reboot, UA_{Reb}), dependability levels *LE*, *Lm* and *LM* and the expected reward rate at steady state, *E[W]*. When necessary, other measures such as Mean Down Time, *MDT* (due to S_{NS} state class), or Mean Time To Failure[2] (*MTTF)* will be evaluated.

4.2 Specific Measures Related to the Distributed Shared Memory

The aim is to evaluate the performance degradation due the loss of memory banks of the distributed shared memory. We define specific measures associated with the sixteen specific failure modes Mem_i (i = {1, 2,...,16}), corresponding to the state classes where i memory banks are lost (cf. Section 3.1.4).

Let q_i denotes the performance index associated with the specific failure mode Mem_i representing, for instance, the decrease of memory performance in that mode due to the loss of i memory banks.

The expected reward rate *E[Wm]* at steady state is given by:

$$E[Wm] = \sum_{i=0,16} q_i\, Prob\{Mem_i\} \qquad (2)$$

where q_0 is the performance index in Mem_0 in which all memory banks are OK.

---

2 Failure refers here to the No Service state class, NS.

5. Dependability Modeling

The dependability model heavily depends on the evaluated measures. However, a model can be built in such a way that several measures can be evaluated from it. In our case, for each system we have built the same model of the selected measures. Moreover, for each system, we built the same architectural model for X and Y even though, the model of Y could be less detailed than the model of X, as the set of considered failure modes is reduced. Using the same model for X and Y reduces the modeling effort.

The models of the reference and extended systems are detailed in [24]. The extended system architectural model differs by an additional node sub-model and the associated switching procedure taking into account the switching coverage factor.

In the architectural model, a sub-model is associated to each component, including controllers, power supply units, buses, diagnostic equipment, etc. When identical components in the same node have the same behavior with respect to the evaluated measures, they are modeled by one sub-model. We thus exploit the symmetry of the components to reduce the state space, from the beginning. This is for example the case of the four processors and the four memory banks within a node.

Due to the large number of components in each node, the reference architecture model is very complex despite the relatively small number of nodes. It is indeed composed of 40 sub-models for X*r* and Y*r* (among which 19 are different). To give an idea about model complexity, the GSPN of the reference system has 183 places, 376 transitions and 82 tokens (in the initial marking) for X*r* (182 places and 351 transitions for Y*r*). The GSPN for the extended system has 216 places, 695 transitions and 97 tokens for X*e* (214 places and 683 transitions for Y*e*). It has been processed using the SURF-2 tool [5].

We illustrate the approach on the GSPN of the four processors of a node in Figure 5 that shows only the main states and transitions.

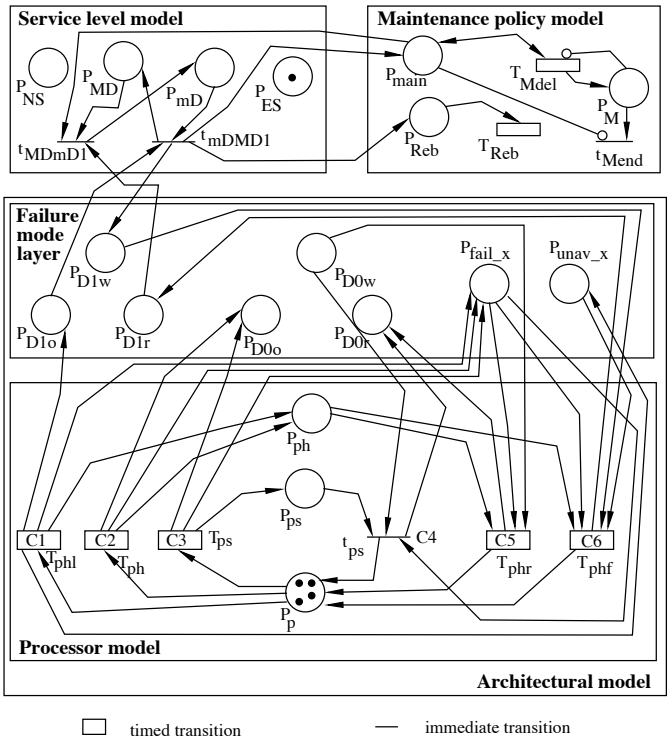
5.1 Model Description

The upper part gives a simplified view of the service-level and maintenance models of X*r*. Places in the left side represent the four service levels and places in the right side represent the maintenance states:

- P_{main}: call for delayed maintenance;
- P_{Reb}: reboot state;
- P_M: the repairman availability.

The lower part of the figure gives the architectural model with the failure mode layer, for X*r* and Y*r*.

For the sake of clarity, not all transitions are shown. In particular, in the architectural model, the enabling conditions C1 to C6 associated with the timed transitions are indicated at the bottom of the figure. Note that these conditions depend on the marking of pace P_{NS} in the service-level model and places P_M and P_{Reb} in the maintenance policy model. Indeed, these places are the only places that are used to enable firing of transitions in the component model directly, without going through the failure mode layer (as stated in Section 2.1.1). More generally, they are the only such places for all the other sub-models of the component model.



Transition	Rate	Definition
T_{ps}	$M(P_p)\ \lambda_{ps}$	software failure of a processor.
T_{phl}	λ_{ph}	hardware failure of the last processor.
t_{ps}	-	one processor becomes available after software failure and reboot.
T_{phr}	μ_{ph}	processor repair after a hardware failure without node isolation.
T_{phf}	μ_{ph}	repair of one processor, reinsertion of the node.
T_{Mdel}	σ_{Mdel}	delayed maintenance.
T_{Reb}	σ_{Reb}	reboot.
t_{Mend}	-	end of maintenance actions (no more failures).
t_{mDMD1}	-	change of service level after the last processor failure.
t_{MDmD1}	-	change of service level after one processor recovery.

C1: $M(P_{ph}) = 3$ & $M(P_{NS}) = M(P_{unav\_x}) = 0$ C4: $M(P_{Reb}) = 1$
C2: $M(P_{ph}) < 3$ & $M(P_{NS}) = M(P_{unav\_x}) = 0$ C5: $M(P_M) = 1$
C3: $M(P_{NS}) = M(P_{unav\_x}) = 0$ C6: $M(P_M) = 1$

Fig. 5. GSPN of the four processors of a node

The failure mode layer gathers all the architecture failure information. Figure 5 shows only information used or updated by the processor model. Each failure mode M ($M \in \{B, DD, DO, D1, D2, D3, C\}$) has 3 places:

- P_{Mo}: occurrence of a failure bringing the system into failure mode M. This failure has to be taken into account in the service-level model;
- P_{Mw}: waiting for repair;
- P_{Mr}: end of repair. The service level has to be updated accordingly.

Also, the interface contains places summarizing the state of each node x:

- P_{fail_x} : number of failures in x;
- P_{unav_x} : when marked, node x is unavailable.

5.2 Interactions Between the Models

We show how the architectural model communicates with service-level and maintenance policy models. In the architectural model, the marking of P_p gives the number of operational processors of node x (its initial marking is 4).

A *hardware failure of a processor* (transition T_{ph}) leads to a minor service degradation level. However, the failure of the last processor (transition T_{phl}) leads to a major service degradation level (because the node becomes unavailable). P_{ps} represents the number of processors with a software failure waiting for reboot and P_{ph} those with a hardware failure waiting for repair.

Let us assume that the last available processor in node x fails. The firing of T_{phl} removes a token from P_p and puts a token in the following places: 1) P_{ph} (there is an additional processor failure in node x), 2) P_{fail_x} (there is an additional failure in node x), 3) P_{unav_x} (node x becomes unavailable) and 4) P_{D1o} (D1 failure mode). The failure is thus recorded in the failure mode layer to update the service-level model. Assuming that the system was in mD, the token from P_{D1o} enables t_{mDMD1}. The firing of t_{mDMD1} changes the current service level from mD to MD and puts a token in P_{D1w} meaning that the failure has been recorded and is waiting for repair. Furthermore, it puts a token in P_{main} for an additional call for maintenance. The presence of a token in P_{main} enables T_{Mdel}, corresponding to the delayed maintenance call. The firing of T_{Mdel} means that a repairman has arrived and the repair is performed. After the repair of a processor of node x, the token is moved from P_{D1w} to P_{D1r} and the service level returns to mD consuming a token from P_{D1r}.

It is worth mentioning that *software failures*, transition T_{ps}, require only a system reboot and do not involve system maintenance.

6. Results

Based on the models of the reference architecture and the extended architecture, as well as on the nominal parameter values, the dependability measures presented in Section 4 have been evaluated and several sensitivity analyses have been carried out. The nominal values of the model parameters (failure rates, repair time, maintenance delay and arrival times and coverage factors) have been either provided by the system manufacturer or assigned according to our experience. Sensitivity analyses have been performed to evaluate their relative influence on dependability and performability measures. For all tables, the bold lines represent the results for nominal parameter values.

The aim of this section is to show the kind of results that could be exploited by the system manufacturer as well as the system end-users. To this end, we have selected a subset of results to show the impact of some parameters. We first give some results related to user X then to user Y (systems X*r*, X*e*, Y*r* and Y*e*), before comparing results for X and Y. Then we show an example of specific measure results. Finally, we present measures related to the clustered system.

6.1 User X

Repair Time. The immediate maintenance time is equal to the sum of the maintenance arrival time and the repair time (cf. Section 3.1.6). The nominal value of each time is 2 hours and it is assumed that all components have the same repair time. To evaluate the impact of maintenance time on dependability, we carry out a sensitivity analysis with respect to repair time.

Table 6 shows that *Lm*, the cumulative time per year spent in S_{mD}, is not sensitive to the repair time; while *LM*, the cumulative time per year spent in S_{MD}, and the unavailability are affected. To a large extent, the unavailability of the system is due to the reboot time, as shown by the last column

of Table 6. The difference between the last two columns gives the unavailability due to S_{NS}. For instance, the reduction by one hour of the repair time reduces the cumulative time per year spent in S_{NS} by 20 min per year (from 34 min to 14 min).

Another way to obtain the cumulative time per year spent in S_{NS} consists in using the MTTF and MDT. Using the nominal values of the parameters, the *MTTF* is 42033 hours (4.8 years) and the *MDT* due to S_{NS} is 2h43, for a repair time if 2 hours. The cumulative time per year spent in S_{NS} is thus equal to 2h43 / 4.8 years = 34 min per year (which is equal to the UA_{NS} given in Table 6 for a repair time of 2 hours).

Table 6. X*r* Dependability measures wrt repair time

Repair time	*LE* (Prob)	*Lm*	*LM*	UA_{NS}	UA_{Reb}	*UA*
5h	0.98866	47h39	48h46	2h08	0h46	2h54
2h	**0.98913**	**47h41**	**46h13**	**0h34**	**0h45**	**1h19**
1h	0.98926	47h42	45h23	0h14	0h44	0h58

Reboot Time. The nominal reboot time is 20 min. Table 6 suggests that, on average, there are two system reboots per year. Table 7 confirms this result for different values of the reboot time. As mentioned in Section 3.1.2, it is assumed that all system reconfigurations force a reboot. In addition, the reinsertion of the off-line repaired component(s) necessitates a system reboot. The results of Tables 6 and 7 argue in favor of performing a reconfiguration without a reboot or with fast reboot (taking less time), whenever possible. Also, Table 7 shows that the 34 min of the unavailability due to S_{NS} are independent from the reboot time.

Table 7. X*r* Dependability measures wrt reboot time

Reboot time	*LE* (pPob)	*Lm*	*LM*	UA_{NS}	UA_{Reb}	*UA*
10 min	0.98916	47h41	46h18	0h34	0h23	0h57
20 min	**0.98913**	**47h41**	**46h13**	**0h34**	**0h45**	**1h19**
30 min	0.98910	47h41	46h09	0h34	1h07	1h41

Maintenance Delay. Immediate maintenance is performed only when the system is in S_{NS}. When the system is in a degraded service state, maintenance is delayed. The nominal maintenance delay is one week for both minor and major service degradation states. Table 8 shows that if the

maintenance delay is two weeks for S_{mD} then *Lm*, the time spent in S_{mD}, is almost multiplied by two. On the other hand, when the delay is reduced to two days for S_{MD}, *LM*, the time spent in S_{MD}, is significantly reduced. In both cases, unavailability is not affected.

Table 8. X*r* Dependability wrt maintenance delay in S_{mD}, S_{MD}

Delay in S_{mD} / S_{MD}	*LE* (Prob)	*Lm*	*LM*	*UA*
2 weeks / 1 week	0.98378	94h33	46h15	1h19
1 week / 1 week	**0.98913**	**47h41**	**46h13**	**1h19**
1 week / 2 days	0.99274	47h51	14h25	1h19
2 days / 2 days	0.99663	13h45	14h24	1h19

Another possibility could be to perform the delayed maintenance in a more regular manner: i. e., to make periodic preventive maintenance without any error reported. However, periodic maintenance could be more expensive than on-request maintenance if the time between two maintenance interventions (i. e., its period) is too short (to improve system availability). A tradeoff has to be made: the periodicity can be optimized through the evaluation of the number of visits to the delayed maintenance states. The models we have developed can be modified to model periodic maintenance: this modification affects only the maintenance part of the model and its interactions with the failure mode layer and service-level model.

Processor Hardware Failure Rate (λ_{ph}). The nominal hardware failure rate of a processor is 10^{-6}/h. Table 9 shows the sensitivity of system dependability to this parameter. If this rate is one order of magnitude lower, the unavailability is reduced by 8 min per year, whereas if the failure rate is one order of magnitude higher, it is increased by 1h10 per year. The most influenced level corresponds to the minor service degradation level. This is due to the presence of 16 processors in the system and the failure of up to 3 processors in the same node leads to minor service degradation.

Table 9. X*r* Dependability wrt λ_{ph}

λ_{ph}	*LE* (Prob)	*Lm*	*LM*	*UA*
10^{-7}	0.99145	28h02	45h40	1h12
10^{-6}	**0.98913**	**47h41**	**46h13**	**1h19**
10^{-5}	0.96602	245h20	49h52	2h29

Remote Cache Controller (RCC) Failure Rate. The nominal failure rate of an RCC is 10^{-7}/h. Table 10 shows that system dependability is affected by the value of this failure rate. Indeed, a failure rate of 10^{-5}/h increases the system unavailability by 15 min per year and doubles the cumulative time per year spent in *LM* (corresponding to the major service degradation level, S_{MD}). The most influenced level is *LM*; this is due to the presence of 4 RCCs and the failure of each of them leads to S_{MD}.

Table 10. X*r* Dependability wrt λ_{RCC}

λ_{RCC}	*LE* (Prob)	*Lm*	*LM*	*UA*
10^{-7}	**0.98913**	**47h41**	**46h13**	**1h19**
10^{-6}	0.98855	47h42	51h13	1h21
10^{-5}	0.98282	47h47	101h10	1h34

It is worth noting that with the failure rate of 10^{-6}/h, we obtain almost the same dependability results as for the nominal value. Hence, choosing a less dependable (and probably less costly) RCC will not much impair the systems dependability. Hence, a tradeoff should be made between the component cost and system dependability improvement.

Spare Node. Table 11 gives the dependability measure for the extended architecture, system X*e*. These results are to be compared with those of Table 6. It can be seen that the system unavailability is unaffected, but *LM*, the time spent in S_{MD}, is reduced by almost 46h per year. The "Spare use" column indicates the time during which the spare is used. For the nominal values, this time is 46h25 per year: it is distributed among S_{mD} and S_{ES}. Note that *LM*, the major service degradation level, corresponds here to the loss of 2 nodes before maintenance achievement. A cumulative time per year spent in S_{MD} of 14 min per year shows that the double failure is unlikely.

Table 11. X*e* Dependability measures

Repair time	*LE* (Prob)	*Lm*	*LM*	UA_{NS}	UA_{Reb}	*UA*	Spare use
2h	**0.99426**	**48h46**	**0h14**	**0h34**	**0h45**	**1h19**	**46h25**
1h	0.99433	48h28	0h14	0h14	0h44	0h58	45h35

Expected Reward Rate. Let us assume that the performance index of a given state class represents

the percentage of system processing capacity in this class. For instance, an index of 0.6 means that the processing capacity is 60%. We have thus $r_{ES} = 1$ and $r_{UA} = 0$. In this context, Table 12 gives the expected reward rate at steady state with various values of the performance indexes r_{mD} and r_{MD}. Obviously, X*e* has higher expected reward rate than X*r*. Note that X*e* is less sensitive to the performance index associated to S_{MD} because of the reduced cumulative time per year spent in S_{MD}. The last line gives system availability.

Table 12. Expected reward rate at steady state, *E[W]*

Performance indexes	*E[W]* for X*r*	*E[W]* for X*e*
$r_{mD} = 0.8$; $r_{MD} = 0.6$	0.996650	0.998726
$r_{mD} = 0.8$; $r_{MD} = 0.7$	0.997178	0.998728
$r_{mD} = 0.9$; $r_{MD} = 0.7$	0.997722	0.999285
$r_{mD} = 0.9$; $r_{MD} = 0.8$	0.998250	0.999288
$r_{mD} = 1$; $r_{MD} = 1$ (A)	0.999849	0.999850

6.2 User Y

For Y, the service is either entire or nil. The availability results according to the repair time are summarized in Table 13. Reducing the repair time improves the availability of the system. This is not surprising since most of the failures lead to immediate maintenance. Note that, as opposed to what was observed for X*r*, system unavailability is mainly due to S_{NS} (difference between the last two columns): 3h25 compared to the 24 min of unavailability due to system reboot.

Considering the nominal values, the *MTTF* is 7175h and the *MDT* due to S_{NS} is 3h09. This means that immediate maintenance is called on average a little bit more than once a year and that the reboot time of 24 min corresponds to a system reboot after maintenance: the system does not exercise reboots in S_{ES} as all failures are tolerated without system reconfiguration. Recall that for X*r*, the immediate maintenance is called on average once every 4.8 years, but the system is rebooted on average twice a year.

The maintenance delay rate does not influence the availability of the system: the maintenance is delayed only for benign failures that do not affect service delivery. Sensitivity analyses with respect to hardware processor and to the RCC failures are similar to those obtained for X*r*.

Table 13. Y*r* Availability wrt the repair time

Repair time	*A* (Prob)	UA_{NS}	UA_{Reb}	*UA*
2h	**0.99957**	**3h25**	**0h24**	**3h49**
1h	0.99970	2h13	0h24	2h37

Spare Node. The presence of a spare node has a direct impact on system availability since the first node failure can be tolerated. Table 14 shows that unavailability is divided by 3 for Y*e* compared to Y*r* (Table 13). Moreover, it can be seen that the unavailability is of the same order of magnitude as for X*r* (these values are 1h19 and 58 min from Table 6 for repair time respectively of 2h and 1h). Considering the nominal values of the parameters, the time during which the spare is used is 93h08. This time is almost equal to the sum of *Lm* and *LM* of Table 6, line 2, that is 93h53. The difference (45 min) corresponds to the time spent in states without an external disk (failure mode DD defined in Table 2).

Table 14. Y*e* Dependability

Repair time	*A* (Prob)	UA_{NS}	UA_{Reb}	*UA*	Spare use
2h	**0.99986**	**0h34**	**0h41**	**1h15**	**93h08**
1h	0.99990	0h15	0h40	0h55	92h17

6.3 Tradeoff Availability - Performance

Columns 3 and 4 of Table 15 report the expected reward rate and availability of the four systems for the nominal values of the parameters (for Y, the expected reward rate is equal to the availability). They show that the reference architecture provides better availability for user X and better performance for Y: there is thus a tradeoff between system performance and availability. On the other hand, the extended architecture provides better availability and better performance for Y. However, in this case, the difference between X and Y availability is not significant. This result is sensitive to the coverage factor *c* (the probability of successful switching from the failing node to the spare one), as shown in columns 5 and 6 that give the same measures for a lower coverage factor (respectively 0.95 and 0.85).

Table 15. Reference and extended architectures

Measure	User	4 nodes	5 nodes	5 nodes	5 nodes
Expected reward rate	X	0.99825	0.99929	0.99924	0.99913
	Y	0.99957	0.99986	0.99984	0.99981
Availability	X	0.99985	0.99985	0.99985	0.99985
	Y	0.99957	0.99986	0.99984	0.99981

6.4 Specific Measures

Table 16 gives the expected reward rate at steady state for various performance index values q_i associated with the sixteen specific failure modes Mem_i selected hypothetically[3], as no experimental values were available when we made the study. It is worth noting that for no memory bank failure states Mem_0, q_0 always equals to 1.

In the three chosen cases, we assumed that the memory service is considered as unavailable when the unavailable memory banks reach 50% of the overall distributed shared memory (i.e., more than 7 unavailable memory banks). The first case is a pure dependability measure giving the system availability (i.e., the system is available as long as more than 7 memory banks are available). The last case is the worst one: the loss of memory banks increases the memory mean access time, thus reducing system performance.

Table 16. Expected reward rate at steady state, *E[Wm]*

Case	Performance indexes	*E[Wm]*
1	$q_i = 1$ i= {1,…,7} $q_k = 0$ k > 7	0.99985
2	$q_i = 0.9$ i= {1-3} $q_j = 0.8$ j= {4-7} $q_k = 0$ k > 7	0.99788
3	$q_i = 0.5$ i= {1-3} $q_j = 0.1$ j= {4-7} $q_k = 0$ k > 7	0.99082

3 q_i represents memory performance decrease. For instance, if the memory mean access time without any memory loss is t_0, it becomes t_i (> t_0) for Mem_i. Then, $q_i = t_0/t_i$ is the performance index corresponding to the resulting memory mean access time increase.

Table 16 shows that the expected reward rate at steady state is slightly influenced by the performance index values: the decrease is 0.9% between the best case (case 1) and the worst case (case 3). This decrease is not important for a degradable highly available system (with A = 99.98% and UA less than 2 hours per year). It may be important for systems with higher unavailability.

6.5 Clustered Systems

As mentioned in Section 3.2, we have considered a clustered system based on the studied reference architecture. Since the clustered systems are composed of four reference architectures, it is possible, under some independence assumptions, to compute the clustered systems performability by combining measures of the reference architecture. We have assumed complete independency between the clusters and evaluated the performability of the clustered system with four clusters. Nevertheless, more complex assumptions can be used[4].

We have considered a set of measures corresponding to the probability in the various failure modes (or groups of failure modes) of Table 3. Table 17 summarizes these measures.

Table 17. Measures for the clustered system

Measure	Definition
LE*c*	entire service possibly with loss of redundancy
Lm*c*	minor degradation where some components are failed but all the clusters still available
LM*c*	major degradation with loss of 1, 2 or 3 clusters
UA*c*	no service (maintenance or system reboot)

For the defined measures, Table 18 shows the impact of P_{CR}, the probability that the common resources (mainly the interconnection ring) are unavailable and P_{RC}, the probability that the ring controller is down.

4 Indeed, we can use the same modeling approach for the whole clustered system as well. However, as the clustered system is composed of four identical systems that are loosely coupled, its performability measures can easily be obtained by combining the performability measures of its components. This is a way of model and result re-use.

Table 18. Dependability of the clustered system

P_{CR}	P_{RC}	L_{ES}	L_{mD}	L_{MD}	L_{NS}
10^{-6}	10^{-6}	0.98913	93h54	1h20	31s
10^{-6}	10^{-5}	0.98912	93h54	1h24	31s
10^{-5}	10^{-6}	0.98912	93h54	1h20	5min15s
10^{-5}	10^{-5}	0.98911	93h54	1h24	5min15s

We notice that the unavailability is very low. This is due to the low probability of the corresponding events: it is scarce to have a common resource failure or the simultaneous unavailability of the four clusters. Also, let us note the huge reduction of system unavailability compared to the reference and extended systems. For the two latter, the lowest unavailability (obtained for the reference system with a repair time of 1 hour) is 58 min / year to be compared to about 5 min / year for the clustered system. Indeed, the clustered system is typically to be used for system requiring high-availability, such as in air traffic control systems or even in web applications for which high availability is important.

It is worth noting the importance of the cumulative time per year spent in Lm*c*, almost 4 days, compared to LM*c*, about 1 hour. In the first case, all the clusters are available with possible degraded performance due to the failure of some components. Whereas in the second case, there is at least one cluster lost, which is unlikely (probability of 1.5×10^{-4} corresponding to 1h19 min of downtime per year, as from Table 7).

Finally, note that P_{RC} mainly influences LM*c* while P_{CR} mainly influences UA*c*. This is obvious as the ring controller (RC) has a local impact while the common resources (CR) such as the interconnection ring have a global impact. The results allow quantification of this impact.

6.6 Summary of Results

The main results presented in this section can be summarized as follows:

User X

- System unavailability is mainly due to the reboot time;
- The No Service states are reached, on average, once each 4.8 years but the system is rebooted twice a year;
- The maintenance delay affects only the cumulative time per year spent in states with minor and major service degradation; while the repair time affects system unavailability;
- The addition of a spare (i.e., the extended system) does not affect system unavailability but reduces the cumulative time per year spent in states with major service degradation.

User Y

- System unavailability is mainly due to the maintenance time.
- The No Service states are reached, on average, once a year and the system is rebooted once a year (following system maintenance).
- The addition of a spare considerably reduces system unavailability.

Clustered system

The availability of clustered systems is very high even compared to the availability of the extended architecture. Indeed, the clustered system is to be used for applications requiring very high-availability.

7. Conclusions

This paper was devoted to the brief presentation of a performability modeling approach and its application to a particular family of multipurpose, multiprocessor systems (MMS). The modeling approach is modular. Its originality is in the separation between the architectural and the environmental concerns. This separation is very important; in particular it allows us to consider a system manufacturer perspective, in which the end-user needs are explicitly accounted for. Also, it

allows evaluation of comprehensive measures defined with respect to service levels and specific measures defined with respect to the architecture specific features.

The considered MMS family is under investigation by a system manufacturer. The results obtained for this MMS can be classified into two categories: those supporting the manufacturer choices and those that will support the potential end-users choices. Of course, these results are not independent and have to be used together. Proper design choices by the manufacturer will — hopefully — be of great benefit for the users.

From the manufacturer perspective, the results are mainly related to:

- the selection of components according to the impact of their failure rates on dependability measures (and their cost most probably);
- the decision concerning the reboot policy;
- the provision of a spare node (a tradeoff should be made between the dependability improvement and the additional difficulty for developing the underlying mechanisms for the insertion of the spare into the system);
- the availability of quantified information that can be used as commercial arguments for future system purchasers;
- the study of impact of the distributed shared memory and more generally of the particular system features;
- the forecasting of clustered architectures given the individual cluster measures.

From the end-user perspective, the results concern:

- the selection of the maintenance policy (delayed or immediate maintenance);
- the choice between the reference architecture and the extended one, and more generally between all available solutions.

Another important point of interest concerns the exploitation by the end-user of the various degradation possibilities offered by the architecture. According to the service expected by the application, the user has to make choices concerning the service degradation levels he/she can

accept and the tradeoff between performance and availability. This choice may affect the architecture of the applicative software.

References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems", *ACM Transactions on Computer Systems*, Vol. 2 (2), pp. 93-122, 1984.

[2] J. Arlat and J.-C. Laprie, "Performance-Related Dependability Evaluation of Supercomputer Systems", Proc. *13th Int. Symp. on Fault-Tolerant Computing (FTCS-13)*, Milano, Italy, 1983, pp. 276-283.

[3] M. Balakrishnan and K. S. Trivedi, "Componentwise Decomposition for an Efficient Reliability Computation of Systems with Repairable Components", Proc. *25th Int. Symp. on Fault-Tolerant Computing (FTCS-25)*, Pasadena, CA, USA, 1995, pp. 259-268.

[4] G. Balbo, S. C. Bruell, and S. Ghanta, "Combining Queuing Networks and GSPNs for the Solution of Complex Models of System Behaviour", *IEEE Transactions on Computers*, Vol. 37 (10), pp. 1251-1268, 1988.

[5] C. Béounes, M. Aguéra, J. Arlat, S. Bachman, C. Bourdeau, J. E. Doucet, K. Kanoun, J.-C. Laprie, S. Metge, J. Moreira de Souza, D. Powell, and P. Spiesser, "SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software systems", Proc. *23rd Int. Symp. on Fault-Tolerant Computing*, Toulouse, France, 1993, pp. 668-673.

[6] C. Betous-Almeida and K. Kanoun, "Dependability Evaluation From Functional to Structural Modelling", Proc. *20th Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP'2001)*, Budapest, Hungary, 2001, pp. 227-237.

[7] A. Bondavalli, I. Mura, S. Chiaradonna, R. Filippini, S. Poli, and F. Sandrini, "DEEM: a Tool for the Dependability Modeling and Evaluation of Multiple Phased Systems", Proc. *Int. Conf. on Dependable Systems and Networks (DSN'2000)*, New York, USA, 2000, pp. 231-236.

[8] A. Bondavalli, I. Mura, and M. Nelli, "Analytical Modeling and Evaluation of Phased Mission Systems in space Applications", Proc. *2nd IEEE High Assurance System Engineering Workshop (HASE)*, Bethesda, MD, USA, 1997, pp. 85-91.

[9] A. Bondavalli, I. Mura, and K. S. Trivedi, "Dependability Modeling and Sensitivity Analysis of Scheduled Maintenance Systems", Proc. *3rd European Dependable Computing Conference (EDCC-3)*, Prague, Czech Republic, 1999, pp. 7-23.

[10] J. Campos, S. Donatelli, and M. Silva, "Structured Solution of Asynchronously Communicating Stochastic Modules", *IEEE Transactions on Software Engineering*, Vol. 25 (2), pp. 147-165, 1999.

[11] G. Ciardo and A. S. Miner, "A data structure for the efficient Kronecker solution of GSPNs", Proc. *8th Int. Workshop on Petri Nets and Performance Models (PNPM'99)*, Zaragoza, Spain, 1999, pp. 22-29.

[12] G. Ciardo, J. Muppala, and K. S. Trivedi, "SPNP: Stochastic Petri Net Package", Proc. *3rd Int. Workshop on Petri Nets and Performance Models (PNPM'89)*, Kyoto, Japan, 1989, pp. 142-151.

[13] G. Ciardo and K. Trivedi, "A Decomposition Approach for Stochastic Reward Net Models", *Performance Evaluation*, Vol. 18 (1), pp. 37-59, 1993.

[14] S. Donatelli, "Superposed Generalized Stochastic Petri net: definition and efficient solution", Proc. *15th Int. Conf. on Applications and Theory of Petri Nets*, 1994, pp. 258-277.

[15] N. Fota, M. Kaâniche, and K. Kanoun, "Dependability Evaluation of an Air Traffic Control Computing System", *Performance Evaluation*, Vol. 35 (3-4), pp. 253-273, 1999.

[16] N. Fota, M. Kâaniche, and K. Kanoun, "Incremental Approach for Building Stochastic Petri Nets for Dependability Modeling", in *Statistical and Probabilistic Models in Reliability*, D. C. Ionescu and N. Limnios, Eds., Birkhäuser, 1999, pp. 321-335.

[17] K. Kanoun and M. Borrel, "Dependability of Fault-Tolerant Systems - Explicit Modeling of the Interactions between Hardware and Software Components", Proc. *Int. Computer Performance & Dependability Symposium (IPDS'96)*, Urbana-Champaign, IL, USA, 1996, pp. 252-261.

[18] K. Kanoun, M. Borrel, T. Moreteveille, and A. Peytavin, "Availability of CAUTRA, a Subset of the French Air Traffic Control System", *IEEE Transactions on Computers*, Vol. 48 (5), pp. 528-535, 1999.

[19] R. Marie and A. Jean-Marie, "Quantitative Evaluation of Discrete Event Systems: Models, Performances and Techniques", Proc. *5th Int. Workshop on Petri Nets and Performance Models (PNPM'93)*, Toulouse, France, 1993, pp. 2-11.

[20] J. F. Meyer, "Performability: A Retrospective and Some Pointers to the Future", *Performance Evaluation*, Vol. 14 (3-4), pp. 139-156, 1992.

[21] J. F. Meyer and W. H. Sanders, "Specification and Construction of Performability Models", Proc. *Int. Workshop on Performability Modeling of Computer and Communication Systems*, Mont Saint Michel, France, 1993, pp. 1-32.

[22] J. K. Muppala, A. Sathaye, R. C. Howe, and K. S. Trivedi, "Dependability Modeling of a Heterogeneous VAX-cluster System Using Stochastic Reward Nets", in *Hardware and Software Fault Tolerance in Parallel Computing Systems*, D. R. Avresky, Ed., Ellis Horwood Ltd., 1992, pp. 33-59.

[23] I. Mura and A. Bondavalli, "Markov Regenerative Stochastic Petri Nets to Model and Evaluate Phased Mission Systems Dependability", *IEEE Transactions on Computers,* Vol. 50(12), pp. 1337-1351, 2001.

[24] M. Rabah, "Dependability Evaluation of Multipurpose Multiprocessor Systems", National Polytechnique Institute, Toulouse, France, PhD Thesis (*in French*), LAAS report 00453, November 2000.

[25] M. Rabah, "Performability of a Distributed Shared Memory Multiprocessor Systems", Proc. *Int. Conf. on Dependable Systems and Networks (DSN'2000)*, New York, NY, USA, 2000, pp. A37-A39.

[26] M. Rabah and K. Kanoun, "Dependability Evaluation of a Distributed Shared Memory Multiprocessor System", Proc. *3rd European Dependable Computing Conference (EDCC-3)*, Prague, Czech Republic, 1999, pp. 42-59.

[27] I. Rojas, "Compositional Construction of SWN Models", *The Computer Journal*, Vol 38(7), pp. 612-621, 1995.

[28] R. A. Sahner and K. S. Trivedi, "Reliability Modeling Using SHARPE", *IEEE Transactions on Reliability*, Vol. R-36 (2), pp. 186-193, 1987.

[29] W. H. Sanders, W. D. Obal II, M. A. Qureshi, and F. K. Widjanarko, "The *UltraSAN* Modeling Environment", *Performance Evaluation*, Vol. 24 (1-2), pp. 89-115, 1995.

[30] V. Santonja, M. Alonso, J. Molero, J. J. Serrano, P. Gil, and R. Ors, "Dependability Models of RAID Using Stochastic Activity Networks", Proc. *2nd European Dependable Computing Conference (EDCC-2)*, Taormina, Italy, 1996, pp. 141-158.

[31] L. A. Tomek, V. Mainkar, R. M. Geist, and K. S. Trivedi, "Reliability Modeling of Life-Critical, Real-Time Systems", *Proceedings of the IEEE, Special Issue on Real-Time Systems*, Vol. 82 (1), pp. 108-121, 1994.

[32] K. S. Trivedi, B. R. Haverkort, A. Rindos, and V. Mainkar, "Techniques and Tools for Reliability and Performance Evaluation: Problems and Perspectives", Proc. *7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Vienna, Austria, 1994, pp. 1-24.

Acknowledgement:

We are grateful to Mohamed Kaâniche and Jean Arlat from LAAS for their relevant comments when reading earlier versions of this paper. Also, we would like to thank the anonymous referees whose comments helped us to make the latest fine-tuning of the paper.

Mourad Rabah is currently at:

Université de La Rochelle, IUT INFO, 15, rue François de Vaux de Foletier,

17026 La Rochelle Cedex

mrabah@univ-lr.fr

Mourad Rabah started his undergraduate studies in computer science at Oran University (Algeria) and finished them in Nantes University (France) by a Bachelor of Science Degree. Then, he obtained a Master of Science degree in computer science from Rennes University (France). Finally, he received his Doctorate from the National Polytechnic Institute of Toulouse (France) prepared at LAAS-CNRS on modeling and dependability evaluation of multipurpose multiprocessor systems. Now, he is an associate professor at University of La Rochelle (France) performing his research at L3I laboratory. His recent research activity is focused on the agreement problems in non fault-tolerant asynchronous distributed systems.

Karama Kanoun is currently Directeur de Recherche at CNRS. Her current research interests include modeling and evaluation of computer system dependability considering hardware as well as software. Domains in which she has authored and co-authored more than ninety papers. She has conducted several research contracts and she has been a consultant for several French companies, the European Space Agency (ESA), Ansaldo and for the International Union of Telecommunications. Besides serving on program committees of international conferences, she served as Program Committee co-Chair of the international Symposium on Software Reliability Engineering (ISSRE'94) and of the International Conference on Dependable Systems and Networks (DSN), in 2000. She served as General Chair of ISSRE'95 and of the 18th International Conference on Computer Safety, Reliability and Security (SAFECOMP'99).

She was Visiting Professor at the University of Illinois, Urbana Champaign, in 1998. She acts as a chair of the French SEE Club 63, working group on "Design and Validation for Dependability" and of the Special Interest Group on Dependability Benchmarking (SiGDeB) of the IFIP WG 10.4 on Dependable Computing and Fault Tolerance.

Karama Kanoun was Guest Editor of a Special Issue on Dependability of Computing Systems of the IEEE Transactions on Computers, January 1998. She is Associate Editor of IEEE Transactions on Reliability.