Reliability Assessment of Multiple-Agent Cooperating Systems

I-Ling Yen, Member IEEE
University of Texas at Dallas, Richardson
Ing-Ray Chen, Member IEEE
Virginia Tech, Falls Church

Key Words — **Real-time system, Reliability analysis, Multiple agent cooperation, Adaptive control, Output acceptability.**

Summary & Conclusions — Multiple agent systems are being increasingly used in real-time applications, safety-critical systems, etc. **Many of these applications have soft real-time constraints and imprecise correctness criteria for the output value. A principal characteristic of such systems is the tradeoff between using an efficient but less accurate strategy and a sophisticated but more time-consuming strategy. This paper develops methods for assessing the reliability of these systems in terms of the timeliness & accuracy of the output, to aid decision making. Two commonly used coordination structures, parallel & pipeline, are used. Some adaptive schemes are proposed, *ie*, strategies can be selected dynamically by agents at the run-time according to the input encountered in order to enhance the reliability of the system. The applicability of this approach is demonstrated by a practical multimedia client-server example.**

1. INTRODUCTION

Along with the advances in parallel & distributed computers, multiple server systems are being increasingly used in various applications, such as manufacturing, defense, traffic coordination, and transaction processing. Many of these applications are *soft* real-time systems in the sense that [2, 6, 9]:

- a non-rigid deadline is associated with the system
- a missed deadline does not necessarily result in the failure of the system.

For example,

- in an automated production line, the delay of a certain part can delay the delivery of the final products; however, the missed deadline only degrades the service level, but does not necessarily cause disastrous failures;
- a transaction processing system for stock exchange has no fixed deadline for responding to a request, yet a reasonably fast response time is anticipated. ◄

The output of many programs is not necessarily binary (either correct or wrong) but can be of various quality levels, for example,

- in an avionics target control system the correctness of the output is not precisely defined as a success or a failure; rather, it is defined as some function of the distance between the strike point and the center of the target;
- in an artificial-intelligence based decision-making system [4, 5] the output decision might not be either correct or wrong; rather, the output might be optimal, suboptimal, marginal, or poor. ◄

A typical characteristic of soft real-time systems is the tradeoff between the accuracy of the output and the time spent in computing the output [2, 3, 5- 8]. With a more sophisticated strategy, a more accurate output can be computed with a longer computation time, which, in some cases, degrades the system reliability since the acceptability of the output in terms of timeliness is degraded. Thus, a reliability assessment must consider the correlation between timeliness & correctness (output accuracy) in order to define & select the optimal strategy.

This paper develops assessment methods for evaluating the reliability of a special soft real-time system that consists of multiple cooperating agents. Two types of coordination structures, parallel & pipeline [11, 12], are considered.

- Parallel structure: Each agent accomplishes a task entirely on its own. It can be designed to allow one server to pick up the task of a failed server and compensate for the unfinished work.
- Pipeline structure: Each task is executed by the agents cooperatively, in a pipeline form. It affords the possibility of one agent being able to compensate immediately for errors committed by another agent. ◄

Section 2 defines the system model, and presents a reliability assessment of a single server system. Section 3 discusses two coordination structures, parallel & pipeline, and the adaptive schemes. Section 4 develops reliability assessment models for multiple server systems and gives examples to illustrate the concepts. Section 5 presents a case study for the reliability assessment of a real-time multimedia system in a client-server setting to demonstrate the applicability of this approach.

Notation

n	number of subtasks & agents
j	index for subtask & agent, $j = 1, 2, \ldots, n$ unless otherwise specified
A_j, T_j	[agent, subtask] j
$h_j(t)$	hardware reliability of A_j
$w_j(t)$	software reliability of T_j (due to program bugs)
s_j	strategy chosen by A_j for executing T_j
$s\#k$	strategy $\#k$, where several strategies are available
T	$\{T_1, T_2, \ldots, T_n\}$: a task decomposed into n T_j, each T_j to be executed by A_j
s	$\{s_1, s_2, \ldots, s_n\}$

$\boldsymbol{A}$	$\{A_1, A_2, \ldots, A_n\}$
p_f	Pr{system failure at an input point selection | no failure at previous input point selections}
R	$1-p_f$: system reliability/acceptability
R_s	R when s is chosen
IQ	input quality, IQ $\in$ [0,1]
IQ_j	IQ to A_j
IQ	$\{IQ_1, IQ_2, \ldots, IQ_n\}$
$g_j(IQ_j)$, $G_j(IQ_j)$	Cdf$\{IQ_j\}$
t	time
t_j	execution time of T_j by A_j
t_{j,s_j}	execution time of A_j when s_j is chosen by A_j
$\boldsymbol{t}$	$\{t_1, t_2, \ldots, t_n\}$
q_s(IQ)	acceptability function of the system output in terms of meeting the system correctness requirement when s is chosen; 1 $\Rightarrow$ completely acceptable; 0 $\Rightarrow$ complete failure
$r(t)$	acceptability function of the system output in terms of meeting the system overall timeliness requirement
q_{j,s_j}(IQ)	acceptability function of the A_j output in terms of meeting its correctness requirement when s_j is chosen
$r_j(t)$	acceptability function of the A_j output in terms of meeting its timeliness requirement
$Q_{j,s_j}(y\|x)$	Pr$\{q_{j,s_j}(x) = y\}$
$F_{j,s_j}(t)$	Cdf{completion time of s_j}
$f_s(t)$, $F_s(t)$	[pdf, Cdf] of completion time for s
g(IQ), G(IQ)	[pdf, Cdf] of IQ.

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

2. DEFINITIONS AND SYSTEM MODEL

Definition

- System reliability for a service request. Pr{system can successfully generate an output for the service request without incurring a software, hardware, correctness-related or timeliness-related failure}. This definition follows that for systems with discrete runs [10]:

$$R(1) = 1-p_f$$

Model

1. The system consists of $\boldsymbol{A}$, working together to accomplish a $\boldsymbol{T}$.

2. The correctness- and timeliness-related failures are addressed by considering the acceptability of the output.

3a. Tasks can be executed by different strategies.

3b. Each task is allowed to select from a pool of strategies.

4. A higher q_s(IQ) occurs when the IQ is higher.

5. $r(t)$ can be a step function for hard real-time constraints, and bell-shaped, exponential, or other curves for soft real-time constraints.

6. Assess R_s for various s. ◄

Consider a system for $n=1$. To complete the task successfully requires the hardware & software to be working during the computation. The reliability of the output depends on its correctness & timeliness. Thus, the system reliability using a chosen s can be formulated as:

$$R_s = \int_0^{\infty} \int_0^{1} h_1(t) \cdot w_1(t) \cdot q_s(\mathrm{IQ}) \cdot r(t)\ dG(\mathrm{IQ})\ dF_s(t). \quad (1)$$

The timeliness requirement is considered in (1) by $r(t)\ dF_s(t)$; the correctness requirement is considered by q_s(IQ) dG(IQ). The time space is integrated over $F_s(t)$ and the input quality space is integrated over $G(IQ)$. The execution time not only affects the acceptability of the output in terms of timeliness but also the hardware/software reliability because the longer the system runs, the more likely it encounters a hardware or software failure. R_s is defined with respect to a particular s whose execution time distribution is independent of G(IQ). The problem is to choose the s that maximizes R_s.

The nature of the 4 types of failures in (1) are instructive. While software & hardware failures can be defined precisely with a certain probability when given a time period, it is less intuitive to connect timeliness-related and correctness-related failures with a probability, because the concept of failure for these latter two failure-types does not have a precise definition [7]. Therefore, if software & hardware failures are excluded from (1), then R_s can be defined as the system-acceptability of the output in terms of meeting the timeliness & correctness criteria specified by the service request. Therefore, when no software or hardware failures are assumed, the terms system-reliability & system-acceptability-of-the-output are equivalent (in terms of timeliness & correctness).

3. COORDINATION STRUCTURES

3.1 Parallel Coordination Structure

Each agent completes the subtask assigned to it all by itself. All agents perform the computation in parallel and communicate with each other to achieve the goal. Various strategies can be selected by an agent for executing a subtask. The selection can be adaptive based on the system status. For different applications, different types of adaptation can be considered. The factors include the number of subtasks, output acceptability, and input quality. With a tight time-constraint and many subtasks to execute, the system can adopt a less accurate but faster strategy in order to guarantee reasonable timing. With a poor output quality, the system can re-execute the subtasks with more accurate strategies when time permits. With a poor input quality, it might be better for the system to use a more time-consuming strategy in order to obtain a reasonably accurate output.

Example 1

An air-traffic coordination system where agents consist of parallel coordinators each of which is assigned a certain region. In the usual case, the objective is to ensure safe travel and to

minimize fuel-consumption and delay. If there are too many planes or if some controllers are down, the system can adapt to operate in a degraded mode where the only concern is to ensure safety. The detection of such a situation could be the smallest distance between any two planes with a trigger point indicating the possibility of a collision. ◄

Example 2

Multiple robots are used to construct the map of a new terrain. The terrain is divided into a set of sectors which are distributed among the robots. If all the robots are active, then a detailed map can be constructed. However, if some robots fail, then the degraded strategy can be used to construct a coarse map of the region. Or a fast strategy can be used first and then continue to construct a more detailed map when the original map is found to be unsatisfactory. A detection function can be established by randomly choosing a small region and obtaining a thorough map to compare with the system map constructed with several levels of accuracies. ◄

Input quality can also be a factor in determining the strategy to be used.

Example 3.

A group of aircraft are sent after a group of moving targets. A pattern recognition technique is used for recognizing the target. On a cloudy day, the target (input picture) might not be easy to recognize and, hence, a more sophisticated strategy needs to be used to get a better recognition. Otherwise, a less accurate but more efficient strategy can be used to allow better timing before the location of the target is lost. ◄

3.2 Pipeline Coordination Structure

Let n agents be organized in a pipeline form such that the output of A_i is the input to A_{i+1}. To make the system fault-tolerant, the agents are designed to be able to take over the functions of their adjacent neighbors. The adaptive selection strategy in a pipeline structure is based on the concept of compensation in which at stage #i, A_i selects a strategy to compensate for the shortcomings of the previous $i-1$ stages. For example, if a longer time has been taken in the previous stages and the deadline is near, then A_i can choose a fast strategy. Or, if the output quality of stage #$(i-1)$ is not satisfactory, a more sophisticated strategy can be used to improve the accuracy; and detection methods for judging the output in terms of correctness & timeliness from each stage, not just the final output, need to be established.

Example 4

An assembly line manufactures some parts, and assembles them into a unit. A slightly defective part might be acceptable but it could increase the assembly time. For example, if two pre-drilled holes are off by a few millimeters, then a more precise positioning might be required to align them. A more sophisticated strategy can be used at the assembly stage to make the products acceptable. Similarly, if the input piece of wood has a hard spot at the location to be drilled, then a more sophisticated strategy can be used during the drilling stage to get a more accurate positioning. ◄

Example 5

Two aircraft are assigned to strike a target. The strategy is that if the first one misses, the second one will try again to accomplish the mission. The second one might have to fly lower to ensure an accurate hit. The cost in this case is the increased risk required to achieve the compensation since there is an increasing danger that the aircraft is shot down. ◄

4. RELIABILITY ASSESSMENT

4.1 Parallel Model

4.1.1 Modeling

$\Pr\{A_j \text{ executes } T_j \text{ using } s_j\}$ is computed by (1) in section 2:

$$R_{j,s_j} = \int_0^\infty \int_0^1 h_j(t_j) \cdot w_j(t_j) \cdot q_{j,s_j}(\mathrm{IQ_j}) \cdot r_j(t_j)\, dG_j(\mathrm{IQ_j})\, dF_{j,s_j}(t_j).$$

For $\boldsymbol{A}$, the hardware & software faults can be considered as s-independent, and the input-quality and execution-time distribution can also be considered as s-independent. Thus, in deriving R_s, we can simply use the product of these terms corresponding to the individual agents. However, the correctness & timeliness of the system output must depend on the correctness & timeliness of individual agents and such s-dependence is often application-dependent. Thus,

$$R_s = \int_0^\infty \cdots \int_0^\infty \int_0^1 \cdots \int_0^1 \left[\prod_{j=1}^{n} h_j(t_j) \cdot w_j(t_j) \right] q_s(\mathbf{IQ}) \cdot r(t)\, dG_1(\mathrm{IQ_1}) \cdots dG_n(\mathrm{IQ_n})\, dF_{1,s_1}(t_1) \cdots dF_{n,s_n}(t_n). \quad (2)$$

In general,

$$q_s(\mathbf{IQ}) = \Phi(q_{1,s_1}(\mathrm{IQ_1}), \ldots, q_{n,s_n}(\mathrm{IQ_n})).$$

Functions for Φ include average, maximum, minimum, and product.

Example 6

A car manufacturing system has n agents in parallel which produce engine pistons. The acceptability of a piston is a function of its diameter. The system acceptability of the correctness of the output is most representative using the average of output acceptability of individual agents; *ie*, Φ can be an average function, and,

$$q_s(\mathbf{IQ}) = \frac{1}{n} \cdot \sum_{j=1}^{n} q_{j,s_j}(\mathrm{IQ_j}).$$ ◄

Example 7

A defense system has n agents collectively protect a sector and needs to destroy all the attacking aircraft. Each agent is assigned to protect against 1 attacker. The system reliability of each individual agent is a binary function: successful (in destroying the attacker) or failed. The system is successful only if all the attackers have been destroyed. The product function is most suitable for describing Φ,

$$q_s(\mathbf{IQ}) = \prod_{j=1}^{n} q_{j,s_j}(\mathrm{IQ}_j). \quad \blacktriangleleft$$

If the real-time constraint is specified for the system, then it is necessary to determine the system execution time. Thus,

$$r(t) = r(\theta(t)),$$

θ can be a maximum, minimum or average function depending on the application.

The real-time constraint can also be specified for each individual agent in some systems. Then the acceptability of the system output in terms of timeliness is defined by that of individual agents. Thus,

$$r(t) = \Psi(r_1(t_1), \ldots, r_n(t_n)).$$

An example of Ψ is the product function, in which case the real-time constraints for each individual subtasks must be satisfied. In a system where both Ψ & Φ are the product function,

$$R_s = \prod_{j=1}^{n} R_{j,s_j}.$$

4.1.2 *Example*

Notation

TU time unit (in which t is expressed in the particular case)
$\delta(\cdot)$ impulse function
I_1, I_2, I_3 [best, intermediate, worst] IQ.

This illustrates the reliability-assessment and strategy-selection in multiple agent systems with parallel coordination structure, for $n=2$. A_1 & A_2 are assigned to execute 2 identical subtasks; s#1 & s#2 can be chosen by each agent to execute its assigned subtask. For A_j, the strategy execution time distribution is:

$$f_{j,1}(t) = \delta(t - 2\mathrm{TU}),\ f_{j,2}(t) = \delta(t - 3\mathrm{TU}),$$

Thus s#1 takes 2 TU to execute, and s#2 takes 3 TU to execute. Then,

$t_j(1) = 2$ TU (when s#1 is chosen by A_j),

$t_j(2) = 3$ TU (when s#2 is chosen by A_j).

s#1 is faster but less sophisticated, while s#2 is slower but more sophisticated.

During the execution time periods, let the hardware & software reliabilities be approximately 1. For either A_j, let a discrete IQ distribution exist:

$$g_j(\mathrm{IQ}=I_1) = 0.3,$$

$$g_j(\mathrm{IQ}=I_2) = 0.4,$$

$$g_j(\mathrm{IQ}=I_3) = 0.3.$$

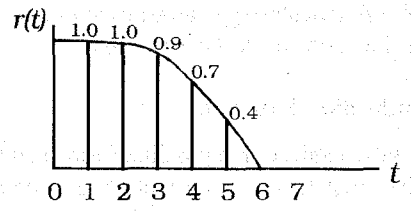
Let $q_s(\mathrm{IQ}) = q_{1,s_1}(\mathrm{IQ}) \cdot q_{2,s_2}(\mathrm{IQ})$.

For either A_j, let $q_{j,1}(\mathrm{IQ})$ & $q_{j,2}(\mathrm{IQ})$ be as in figure 1b. This models s#2 as being more sophisticated and taking more time to execute than s#1, and can provide a higher quality output to meet the system quality acceptability requirement.

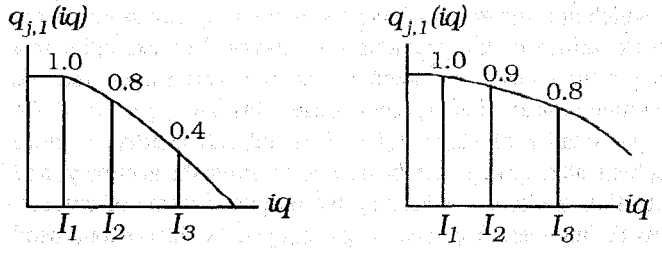
Let $r_1(t) = r_2(t)$ as in figure 1a, and let,

$$r(t_1,t_2) = r_1(t_1) \cdot r_2(t_2).$$

$r(t)$ shows that s#1, which takes 2 TU to execute, can meet the real-time requirement better than s#2, which takes 3 TU to execute; *ie*, $r(t=2\mathrm{TU})$ returns 1 while $r(t=3\mathrm{TU})$ returns 0.9.



(a) $r_1(t)$ and $r_2(t)$



(b) $q_{j,s_j}(iq)$

Figure 1. Parallel Coordination Agent Example

Case A

Two parallel servers execute their tasks s-independently without using any adaptive scheme. To adapt (2), rewrite it in the discrete form:

$$R_s = \prod_{j=1}^{2} \left[\sum_{k=1}^{3} g_j(\mathrm{IQ}=I_k) \cdot [q_{j,s_j}(\mathrm{IQ}=I_k) \cdot r_j(t_{j,s_j})] \right]. \quad (3)$$

The system reliability values using various combinations of strategies chosen by the two agents can be calculated from (3) and are given in table 1.

Table 1. System Reliabilities Under Non-Adaptive, Parallel Structures

Strategies	R_s
$s_1=s\#1,\ s_2=s\#1$	0.5329
$s_1=s\#1,\ s_2=s\#2$	0.5913
$s_1=s\#2,\ s_2=s\#1$	0.5913
$s_1=s\#2,\ s_2=s\#2$	0.6561

Using *s#2* (more sophisticated but less efficient) for A_1 & A_2 gives a better system reliability.

Case B

Consider adaptive schemes where strategies are selected according to the IQ. Under this scheme, adaptive conditions for maximizing the system reliability are statically analyzed. Then, based on the conditions identified, the cooperating agents dynamically select the best strategies to execute according to the IQ at the run time.

Let 4 adaptive schemes be identified: $\rho=0$, $\rho=1$, $\rho=2$, $\rho=3$. The schemes, and the resulting system reliabilities are given in table 2.

Table 2. System Reliabilities under Adaptive, Parallel Structures

Scheme	Strategy	R_s
$\rho=0$	$s_1=s\#2,\ s_2=s\#2$	0.6561
$\rho=1$	if IQ = I_1	
	then $s_1=s_2=s\#1$	
	else $s_1=s_2=s\#2$	0.7056
$\rho=2$	if (IQ = I_1 or I_2)	
	then $s_1=s_2=s\#1$	
	else $s_1=s_2=s\#2$	0.6989
$\rho=3$	$s_1=s_2=s\#1$	0.5329

Eq (3) is again being used for calculating the system reliability values in table 2, except that for $\rho=1$ & $\rho=2$, s_j changes dynamically as $g_j(\mathrm{IQ}=I_k)$ changes.

If s#2 is used when IQ is bad, and s#1 is used when IQ is good (using scheme $\rho=1$ or $\rho=2$), then the system reliability is the best. There is an appreciable improvement due to the use of adaptive strategy schemes.

4.2 Pipeline Model

Notation

$t^*_{k,s}$ $\sum_{j=1}^{k} t_{j,s_j}$: total time to finish stages #1 through #k

$F^*_k(t)$ $\mathrm{Cdf}\{t^*_{k,s}\}$.

4.2.1 Modeling

Consider n stages; stage *#j* is processed by A_j. Each agent can choose 1 strategy from s to accomplish its subtask. The relation between cooperating agents in this structure lies in the input-output relationship between two consecutive stages in the pipeline. A_{j+1} can compensate for the output produced by A_j (both in time & quality) by selecting a strategy such the total real-time and the quality requirements of the task can be best satisfied.

$F^*_k(t)$ is the convolution function of $F_{i,s_i}(t)$ for $i=1,\ldots,k$. Since A_j should be alive until the end of stage *#j*, the hardware reliability of A_j needs to consider $t^*_{j,s}$. On the other hand, the software reliability of stage *#j* need only consider the duration during which the software is active and, hence, need only consider t_{j,s_j}. Thus,

$$R_s = \left[\int_0^\infty r(t^*_{n,s})\ dF^*_n(t^*_{n,s}) \right] \cdot \left[\int_0^1 q_s(\mathrm{IQ}_1) \cdot g_1(\mathrm{IQ}_1)\ d\mathrm{IQ}_1 \right] \cdot \prod_{j=1}^{n} \left[\int_0^\infty h_j(t^*_{j,s})\ dF^*_j(t^*_{j,s}) \cdot \int_0^\infty w_j(t_{j,s_j})\ dF_{j,s_j}(t_{j,s_j}) \right]. \quad (4)$$

Term #1 accounts for the acceptability of the output in terms of its timeliness; term #2 accounts the acceptability of the output in terms of its correctness; term #3 accounts for the hardware/software reliability. The $t^*_{n,s}$ in term #1 is the total time required for finishing all the stages in the pipeline. The $q_s(\mathrm{IQ}_1)$ in term #2 is the acceptability function of the system, given $g_1(\mathrm{IQ}_1)$ to stage #1. Term #2 can also be written as:

$$\int_0^1 q_{n,s_n}(\mathrm{IQ}_n) \cdot g_n(\mathrm{IQ}_n)\ d\mathrm{IQ}_n.$$

To evaluate term #2, therefore, we need a way to relate $g_n(\mathrm{IQ}_n)$ with $g_1(\mathrm{IQ}_1)$.

Since the output of stage *#j* is the input of stage *#(j+1)*, we can use the quality acceptability function of stage *#j* to derive the IQ distribution of stage *#(j+1)*. The IQ distribution of stage *#j+1*, $1 \le j < n$, should be expressed as a conditional distribution of the acceptability function of stage *#j* given the IQ of stage *#j*:

$$g_{j+1}(\mathrm{IQ}_{j+1}) = \int_0^1 Q_{j,s_j}(\mathrm{IQ}_{j+1} | \mathrm{IQ}_j) \cdot g_j(\mathrm{IQ}_j)\ d(\mathrm{IQ}_j)$$

$$= \int_0^1 Q_{j,s_j}(\mathrm{IQ}_{j+1}|\mathrm{IQ}_j) \cdot \int_0^1 Q_{j-1,s_{j-1}}(\mathrm{IQ_j}|\mathrm{IQ}_{j-1})$$

$$\ldots \int_0^1 Q_{1,s_1}(\mathrm{IQ}_2|\mathrm{IQ}_1) \cdot g_1(\mathrm{IQ}_1)\ d(\mathrm{IQ}_1)\ \ldots\ d(\mathrm{IQ}_j). \quad (5)$$

Eq (5) can be applied recursively: $g_{j,s_j}(\mathrm{IQ}_j)$ is known, then $g_{j+1}(\mathrm{IQ}_{j+1})$ can be computed using (5). Eq (5) thus helps to derive the distribution of $g_n(\mathrm{IQ}_n)$ when, given $g_1(\mathrm{IQ}_1)$, as long as we are given $Q_{j,s_j}(\mathrm{IQ}_{j+1}|\mathrm{IQ}_j)$, $1 \le j < n$. Term #2 in (4) can be rewritten as:

$$\int_0^1 q_s(\mathrm{IQ}_1) \cdot g_1(\mathrm{IQ}_1)\ d(\mathrm{IQ}_1)$$

$$= \int_0^1 q_{n,sn}(\mathrm{IQ}_n) \cdot g_n(\mathrm{IQ}_n)\ d(\mathrm{IQ}_n)$$

$$= \int_0^1 q_{n,s_n}(\mathrm{IQ}_n) \cdot \int_0^1 Q_{n-1,s_{n-1}}(\mathrm{IQ}_n|\mathrm{IQ}_{n-1})$$

$$\ldots \cdot \int_0^1 Q_{1,s_1}(\mathrm{IQ}_2|\mathrm{IQ}_1) \cdot g_1(\mathrm{IQ}_1)\ d(\mathrm{IQ}_1)\ \ldots\ d(\mathrm{IQ}_n). \quad (6)$$

Thus, the system acceptability of the output in terms of meeting its quality correctness requirement can be evaluated, when $g_1(\mathrm{IQ}_1)$, $Q_{j,s_j}(\mathrm{IQ}_{j+1}|\mathrm{IQ}_j)$, $1 \le j < n$, and $q_{n,s_n}(\mathrm{IQ}_n)$ are known.

4.2.2 Example

Notation

I_1, I_2, I_3 [best, intermediate, worst] IQ
I'_k level k of IQ to stage #2, k = 1,2,3.

A 2-stage pipeline system (n=2) has stage #j being executed by A_j. Each agent can select either s#1 or s#2 to execute the subtask. The pdf of the execution times of the s#k at stages #1 & #2 are:

$$f_{1,1}(t) = \delta(t - 2\mathrm{TU}),\ f_{1,2}(t) = \delta(t - 3\mathrm{TU});$$

$$f_{2,1}(t) = \delta(t - 1\mathrm{TU}),\ f_{2,2}(t) = \delta(t - 2\mathrm{TU}).$$

In each stage, s#1 executes faster than s#2 but is less sophisticated. The hardware & software reliabilities of the two stages are approximately 1 for $t \le 5$.

$$g_1(\mathrm{IQ}=I_1) = 0.3,$$

$$g_1(\mathrm{IQ}=I_2) = 0.4,$$

$$g_1(\mathrm{IQ}=I_3) = 0.3.$$

The acceptability of the output in terms of $r(t)$ is given in figure 2a.

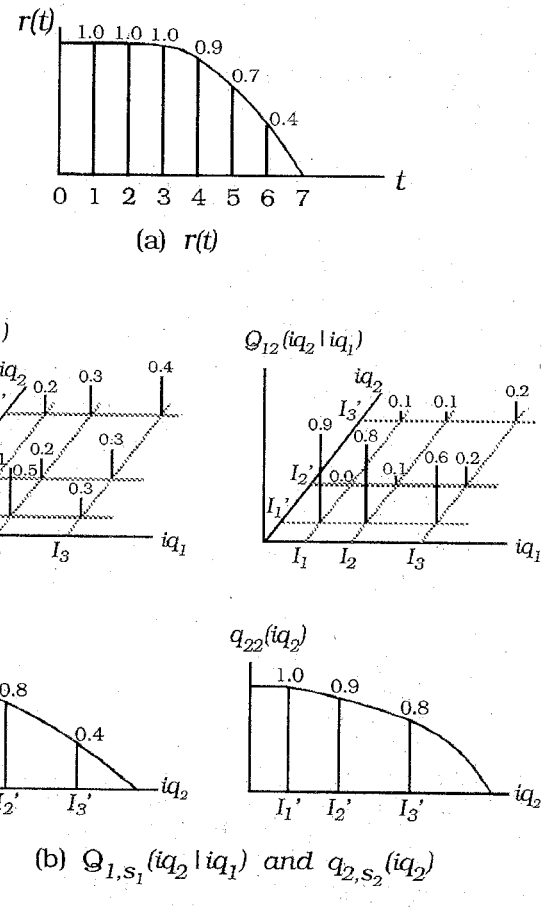


Figure 2. Pipeline Coordination Agent Example

The acceptability functions concerning the correctness of the output should be expressed in terms of the IQ of the current and next stages. Since stage #2 is the last stage, it does not have a next stage and the acceptability function is the same as for a single agent. Thus, there are:

$$Q_{1,1}(\mathrm{IQ}_2|\mathrm{IQ}_1),\ Q_{1,2}(\mathrm{IQ}_2|\mathrm{IQ}_1);$$

$$q_{2,1}(\mathrm{IQ}_2),\ q_{2,2}(\mathrm{IQ}_2);$$

they are given in figure 2b.

It is required that:

$$\sum_{k=1}^{3} Q_{1,s_j}(\mathrm{IQ}_2 = I'_k|\mathrm{IQ}_1 = I_x) = 1, \text{ for } s_j = s\#1, s\#2,$$

and x=1,2,3.

Case A

There are no adaptive schemes. Table 3 gives system reliability values using various combinations of strategies.

Table 3. System Reliabilities Under Non-Adaptive, Pipeline Structures

Strategies	R_s
$s_1=s\#1$, $s_2=s\#1$	0.7800
$s_1=s\#1$, $s_2=s\#2$	0.8010
$s_1=s\#2$, $s_2=s\#1$	0.8118
$s_1=s\#2$, $s_2=s\#2$	0.6657

The best choice of s is ($s_1=s\#2$, $s_2=s\#1$): using $s\#2$ for stage #1 and $s\#1$ for stage #2. ◄

Case B

The schemes are adaptive: strategies are selected according to the IQ. The compensation is done in both stages. Essentially, stage #1 compensates for the actual IQ, and stage #2 compensates for the poor output generated from stage #1. Let 16 adaptive scheme be identified: ρ = 00, 01, 02, 03, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33. The system reliability values using various schemes are calculated from (4) & (6) and given in table 4.

The best choice (marked with † in table 4) is to use $s\#1$ (less sophisticated but quicker) for stage #1 for all IQ and compensate for the previous stage in stage #2 using $s\#1$ when IQ=I'_1 and using $s\#2$ otherwise (ρ = 31). ◄

5. RELIABILITY ASSESSMENT OF A REAL-TIME MULTIMEDIA SYSTEM

Acronyms

GOP group of pictures
I frame uncompressed frame
P frame compressed frame, and can be compensated relative to a preceding *I* frame
B frame compressed frame, and can be compensated relative to 2 neighboring *I* and/or *P* frames
QoS quality of service.

Notation

I'_k level of IQ to stage #2, k = 1,2,3
I'_1 input has good-resolution picture quality and more frequent *I* frames
I'_2 input has either good-resolution but less frequent *I* frames, or more frequent *I* frames but bad-resolution
I'_3 input has bad-resolution and less frequent *I* frames.

This example illustrates the concepts in section 4, and fits the pipeline model. Consider a client-server multimedia application for sending and displaying video frames, with the goal of maximizing the acceptability metric of the system in terms of meeting both the quality & real-time requirements. The server process represents the stage-#1 agent responsible for compressing and sending video frames over the network, while the client process represents the stage-#2 agent responsible for decompressing & displaying the video frames to the end user. For simplicity, classify the IQ of the video frames to the stage-#1 agent into two types:

- good-resolution (I_1),
- bad-resolution (I_2).

Table 4. System Reliabilities under Adaptive, Pipeline Structures

Scheme	Strategies	R_s
ρ = 00	$s_1=s\#2$; $s_2=s\#2$	0.6657
ρ = 01	$s_1=s\#2$; if $IQ_2 = I'_1$ then $s_2=s\#1$, else $s_2=s\#2$	0.8197
ρ = 02	$s_1=s\#2$; if ($IQ_2 = I'_1$ or $IQ_2=I'_2$) then $s_2=s\#1$, else $s_2=s\#2$	0.8287
ρ = 03	$s_1=s\#2$, $s_2=s\#1$	0.8118
ρ = 10	if $IQ_1 = I_1$ then $s_1=s\#1$, else $s_1=s\#2$; $s_2=s\#2$	0.7131
ρ = 11	if $IQ_1 = I_1$ then $s_1=s\#1$, else $s_1=s\#2$; if $IQ_2 = I'_1$ then $s_2=s\#1$, else $s_2=s\#2$	0.8341
ρ = 12	if $IQ_1 = I_1$ then $s_1=s\#1$, else $s_1=s\#2$; if ($IQ_2 = I'_1$ or $IQ_2=I'_2$) then $s_2=s\#1$, else $s_2=s\#2$	0.8428
ρ = 13	if $IQ_1 = I_1$ then $s_1=s\#1$, else $s_1=s\#2$; $s_2=s\#1$	0.8160
ρ = 20	if ($IQ_1 = I_1$ or $IQ_1 = I_2$) then $s_1=s\#1$, else $s_1=s\#2$; $s_2=s\#2$	0.7647
ρ = 21	if ($IQ_1 = I_1$ or $IQ_1 = I_2$) then $s_1=s\#1$, else $s_1=s\#2$; if $IQ_2 = I'_1$ then $s_2=s\#1$, else $s_2=s\#2$	0.8417
ρ = 22	if ($IQ_1 = I_1$ or $IQ_1 = I_2$) then $s_1=s\#1$, else $s_1=s\#2$; if ($IQ_2 = I'_1$ or $IQ_2=I'_2$) then $s_2=s\#1$, else $s_2=s\#2$	0.8460
ρ = 23	if ($IQ_1 = I_1$ or $IQ_1 = I_2$) then $s_1=s\#1$, else $s_1=s\#2$; $s_2=s\#1$	0.7968
ρ = 30	$s_1=s\#1$; $s_2=s\#2$	0.8010
ρ = 31	$s_1=s\#1$; if $IQ_2 = I'_1$ then $s_2=s\#1$, else $s_2=s\#2$	0.8510†
ρ = 32	$s_1=s\#1$; if ($IQ_2 = I'_1$ or $IQ_2=I'_2$) then $s_2=s\#1$, else $s_2=s\#2$	0.8490
ρ = 33	$s_1=s\#1$; $s_2=s\#1$	0.7800

Let the server store a large amount of video files with various quality levels; each video file stores only one type of (uncompressed) video frames. Although the server knows the probability distribution of each type of video file, it does not know *a priori* which type of video file will be requested by the user during the run time. The IQ for stage #1 are discrete:

$$g_1(\mathrm{IQ}=\mathrm{I}_1) = 0.9,\ g_1(\mathrm{IQ}=I_2) = 0.1.$$

In Moving Picture Experts Group (MPEG) compression technology, a GOP consists of a fixed sequence of frames beginning with an *I* frame. Other frames in the same GOP can be *I*, *P*, or *B* frames. Based on the number & types of frames put in a GOP, various GOP formation strategies can be established. Various degrees of compression can be applied to the *P* & *B* frames in the GOP. Consider 2 strategies for the server agent (at stage #1 in the pipeline structure):

- *s*#1 compresses the video data into 15 frames per GOP, consisting of 1 *I* frame, 4 *P* frames, and 10 *B* frames (IBBPBBPBBPBBPBB); a higher compression mode is set to compress the *P* & *B* frames. The *s*#1 yields more compressed video data, and requires less data processing & transferring time, however, it can degrade the quality of the video frames since more frames are in compressed form and must be recovered by the client agent.
- *s*#2 compresses the video data into 9 frames per GOP, consisting of 1 *I* frame, 2 *P* frames, and 6 *B* frames (IBBPBBPBB); a lower compression mode is set to compress the *P* & *B* frames. The *s*#2 requires more data processing & transferring time, and thus can better retain the original picture quality. ◄

The execution time of the server agent includes two parts:

- compress the video frames,
- send the data through the network.

(We ignore other execution time factors such as the time for the server to process the request.) Due to the system load and the network traffic, the execution time in realistic systems are modeled as probability distributions. The execution time varies frame by frame due to the degree of compression and other factors. However, for illustration here, the execution times are modeled as impulse functions which represent the average execution time for processing & transferring each individual frame.

The execution-time strategy pdf's at stage #1 are:

$$f_{1,1}(t) = \delta(t - 20\mathrm{ms}),\ f_{1,2}(t) = \delta(t - 38\mathrm{ms}).$$

When the compressed frames arrive at stage #2, the client agent attempts to recover them, and then displays the decompressed frames to the end user. The client agent can use various strategies to recover the compressed frames in a GOP. Consider 2 strategies for the client agent:

- *s*#1 uses a less accurate decompression method and carries out motion compensation to within 1-pixel resolution;
- *s*#2 uses a more accurate decompression method and performs motion compensation to within ½-pixel resolution. ◄

Both *s*#1 & *s*#2 involve prediction or interpolation methods for motion compensation [1]. The execution-time strategies pdf's (per frame) at stage #2 are:

$$f_{2,1}(t) = \delta(t - 18\mathrm{ms}),\ f_{2,2}(t) = \delta(t - 35\mathrm{ms}).$$

s#1 executes faster than *s*#2, but *s*#1 can degrade the picture resolution more.

The IQ to stage #2 can be classified in terms of the resolution of the pictures and the frequency of the *I*frames received from stage #1. The more frequently the *I*frames are received by stage #2, the better the picture quality can be recovered relative to the original resolution of the pictures.

Based on the 3 IQ levels for IQ_2, estimate $Q_{1,1}(\mathrm{IQ}_2|\mathrm{IQ}_1)$, $Q_{1,2}(\mathrm{IQ}_2|\mathrm{IQ}_1)$ as follows.

a. *s*#1 in stage #1:

$$Q_{1,1}(\mathrm{IQ}_2=I'_1|\mathrm{IQ}_1=I_1) = 0,$$

$$Q_{1,1}(\mathrm{IQ}_2=I'_2|\mathrm{IQ}_1=I_1) = 0.95,$$

$$Q_{1,1}(\mathrm{IQ}_2=I'_3|\mathrm{IQ}_1=I_1) = 0.05,$$

$$Q_{1,1}(\mathrm{IQ}_2=I'_1|\mathrm{IQ}_1=I_2) = 0,$$

$$Q_{1,1}(\mathrm{IQ}_2=I'_2|\mathrm{IQ}_1=I_2) = 0,$$

$$Q_{1,1}(\mathrm{IQ}_2=\mathrm{I}'_3|\mathrm{IQ}_1=I_2) = 1;$$

b. *s*#2 in stage #1:

$$Q_{1,2}(\mathrm{IQ}_2=I'_1|\mathrm{IQ}_1=I_1) = 0.99,$$

$$Q_{1,2}(\mathrm{IQ}_2=I'_2|\mathrm{IQ}_1=I_1) = 0.01,$$

$$Q_{1,2}(\mathrm{IQ}_2=I'_3|\mathrm{IQ}_1=I_1) = 0,$$

$$Q_{1,2}(\mathrm{IQ}_2=I'_1|\mathrm{IQ}_1=I_2) = 0,$$

$$Q_{1,2}(\mathrm{IQ}_2=I'_2|\mathrm{IQ}_1=I_2) = 0.95,$$

$$Q_{1,2}(\mathrm{IQ}_2=I'_3|\mathrm{IQ}_1=I_2) = 0.05.$$

$Q_{1,s_1}(\mathrm{IQ}_2|\mathrm{IQ}_1)$ defines the probability distribution of IQ_2 when given the 'knowledge of the input distributions of IQ_1' and 'knowledge that s_1 (which can be either *s*#1 or *s*#2) is being used by stage #1. The numbers for $Q_{1,1}(\mathrm{IQ}_2|\mathrm{IQ}_1)$ & $Q_{1,2}(\mathrm{IQ}_2|\mathrm{IQ}_1)$ were estimated based on the relationship of the IQ and the strategy under consideration. For example, consider using *s*#2 in stage #1. If $\mathrm{IQ}_1 = I_1$ (good-resolution), it was estimated that, with probability 0.99, that the input to stage #2 (output of stage #1) will be of quality I'_1, $Q_{1,2}(\mathrm{IQ}_2=I'_1|\mathrm{IQ}_1=I_1) = 0.99$, because the frames received by stage #2 are likely (estimated to be 0.99) to retain good-resolution and will also

contain more frequent I frames when $s\#2$ is in action in stage #1 and also when the picture IQ to stage #1 is known to be of good-resolution. On the other hand, $Q_{1,2}(\mathrm{IQ}_2 = I'_3 | \mathrm{IQ}_1 = I_1) = 0$, because it is impossible to have a bad-resolution input to stage #2 when $s\#2$ is in action in stage #1 and also when the input to stage #1 is known to be of good-resolution.

In a multimedia client-server system, the end user at the client side can specify the QoS requirements in terms of:

a. the quality of the display desired,
b. the real-time playback requirement.

The system then can derive the quality and real-time requirement acceptability functions based on the QoS requirements specified by the end user. The acceptability of the output quality with respect to the QoS specified by the end user can be expressed in terms of $q_{2,s_2}(\mathrm{IQ}_2)$. Based on the QoS specified by the end user, the $q_{2,1}(\mathrm{IQ}_2)$ and $q_{2,2}(\mathrm{IQ}_2)$ are derived and shown in figures 3b & 3c, for $s\#1$ & $s\#2$ being used in stage #2, respectively. These two functions reflect the fact that $s\#1$ in stage #2 yields a worse output quality compared with $s\#2$, since it uses a faster but less accurate way to recover the compressed frames. Thus, the acceptability of the display quality under $s\#1$ is lower when compared with $s\#2$ for the same IQ to stage #2.

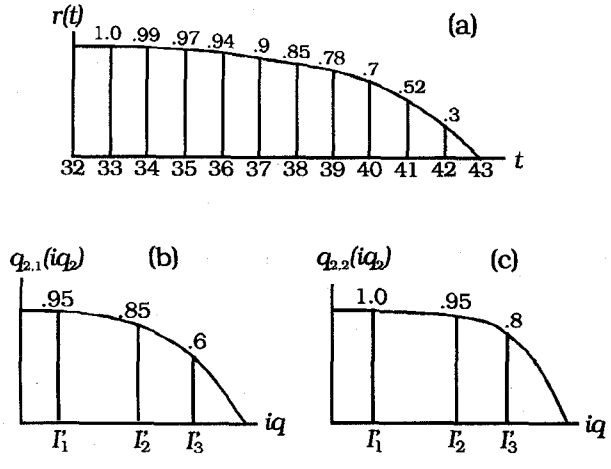


Figure 3. Multimedia System Example

The end user at the client side can also specify the desired real-time requirement as part of its QoS requirement. Based on the real-time requirement, the acceptability function of the output in terms of $r(t)$ can be derived. We consider the execution times of individual frames. Let the end user desire a playback speed of 30 frames/second; then the client agent should deliver 1 frame in every 33ms. Typically both the server & client agents can provide buffering to accommodate load fluctuation and network traffic delay. Thus the constraint on the execution time per frame can be modeled as a soft real-time requirement curve. For execution time (per frame) of 33ms or less, $r(t) = 1$. If the processing time exceeds 43ms, then $r(t) = 0$ (not acceptable with respect to the end user QoS requirement). Figure 3a shows the curve for $r(t)$. The acceptability levels for 33ms $< t <$ 42ms have a decreasing curve with $0 < r(t) < 1$. The two stages can process the frames concurrently in a pipeline mode. Thus, a maximum function is applied to the execution times of the two stages to yield the total execution time.

Compute the system reliability using (4) & (6). The system reliability values using various combinations of strategies are calculated and given in table 5. Let the software & hardware reliabilities be approximately 1.

Table 5. System Reliabilities of the Multimedia System Under the Pipeline Structure

Strategies	R_s
$s_1 = s\#1$, $s_2 = s\#1$	0.8137
$s_1 = s\#1$, $s_2 = s\#2$	0.9004
$s_1 = s\#2$, $s_2 = s\#1$	0.7971
$s_1 = s\#2$, $s_2 = s\#2$	0.8447

The best choice of s is ($s_1 = s\#1$, $s_2 = s\#2$) wherein the system can maximize R_s to about 90%: 90% satisfaction with respect to the end user's specified QoS in terms of meeting both the quality and real-time display requirements.

ACKNOWLEDGMENT

This work is supported in part by US NSF under Grant No. CCR-9521419 and CCR-9634249, and also by the ROC National Science Council under grant NSC-85-2213-E-006-069 and NSC-86-2745-E-006-020.

REFERENCES

[1] P.K. Anddleigh, K. Thakrar, *Multimedia Systems Design*, 1994; Prentice Hall.
[2] F.B. Bastani, E. Leiss, "On the overall reliability of hardware/software systems", *Proc. FJCC*, 1987 Oct, pp. 528-533; Dallas, TX.
[3] M. Boddy, "Anytime problem solving using dynamic programming", *9th National Conf. Artificial Intelligence*, 1991, pp 738-743.
[4] I.R. Chen, F.B. Bastani, "Effect of artificial-intelligence planning procedures on system reliability", *IEEE Trans. Reliability*, vol 40, 1991 Aug, pp 364-369.
[5] I.R. Chen, F.B. Bastani, T.W. Tsao, "On the reliability of AI planning software in real-time applications", *IEEE Trans Knowledge & Data Eng'g*, vol 7, 1995 Feb, pp 4-13.
[6] J.Y. Chung, J.W.S. Liu, K.J. Lin, "Scheduling periodic jobs that allow imprecise computations", *IEEE Trans. Computers*, vol 39, 1990 Sep, pp 1156-1174.
[7] V. Grassi, C. Antonelli, "Optimal design of fault-tolerant soft-real-time systems with imprecise computations" *Proc. Dependable Computing - EDCC-1* (K. Echtle, D. Hammer, D. Powell, *Eds*), 1994 LNCS 852; Springer Verlag.
[8] B. Hamidzadeh, S. Shekhar, "Specification and analysis of real-time problem solvers", *IEEE Trans. Software Eng'g*, vol 19, 1993 Aug, pp 788-803.
[9] E.D. Jensen, "Asynchronous decentralized real-time computer systems", *Proc. NATO Adv. Study Inst. on Real-Time Computing*, 1992 Oct.

[10] J.-C. Laprie, K. Kanoun, "Software reliability and system reliability", *Hdbk. Software Reliability Engineering* (M.R. Lyu, *Ed*), 1996, p 36; IEEE Computer Society Press.

[11] I.L. Yen, F.B. Bastani, "Robust coordination in distributed multi-server systems", *Proc. IEEE Workshop Adv. in Parallel & Distributed Systems*, 1993, pp 133-138; Princeton.

[12] I.L. Yen, F.B. Bastani, "Systematic incorporation of efficient fault tolerance in systems of cooperating parallel programs", *Int'l Symp. Fault-Tolerant Computing*, 1994 Jun, pp 154-163; Austin, TX.

AUTHORS

Dr. I-Ling Yen; Computer Science; Univ. of Texas at Dallas; Richardson, Texas 75083-0688 USA.
Internet (e-mail): yen@cps.msu.edu

I-Ling Yen received her BS from Tsing-Hua University, Taiwan, and her MS & PhD in Computer Science from the University of Houston, University Park. She is an Associate Professor of Computer Science at the University of Texas at Dallas. Dr. Yen's research interest is on parallel & distributed processing and fault-tolerant computing. She has developed several efficient fault-tolerance methods, including inherent fault tolerance, multilevel robust data structures, and repetitive fault tolerance for parallel & distributed applications. She is also conducting research in reliability assessment for multi-agent systems and self-stabilizing systems. Dr. Yen is a member of the IEEE Computer Society.

Dr. Ing-Ray Chen; Computer Science Dep't; Virginia Tech, Northern Virginia Ctr; 7054 Haycock Road; Falls Church, Virginia 22043 USA.
Internet (e-mail): irchen@vt.edu

Ing-Ray Chen received his BS from National Taiwan University, Taipei, and his MS & PhD in Computer Science from the University of Houston, University Park. He is an Associate Professor in the Department of Computer Science at Virginia Tech. His research interests are in reliability and performance analysis, and real-time intelligent systems. Dr. Chen is a member of the IEEE Computer Society and the ACM.

Manuscript TR95-107 received 1995 July 11; revised 1996 November 20

Responsible editor: R.J. Loomis

Publisher Item Identifier S 0018-9529(97)04562-4 ◄TR►

CALL FOR PAPERS CALL FOR PAPERS CALL FOR PAPERS CALL FOR PAPERS CALL FOR PAPERS CALL FOR PAPERS

Special Supplement for the IEEE Reliability Society — 50th Anniversary

The year **1999** is the 50th anniversary of the IEEE Reliability Society — formerly the IRE (Institute of Radio Engineers) Professional Group on Reliability and Quality Control. To mark the occasion, a special supplement for the 1998 December issue will be devoted to the Society's history. Articles desired are historical perspectives on reliability specialties such as:

- Electronics reliability,
- Mathematical theory,
- Reliability education,
- Space reliability technology,
- Nuclear power-plant reliability,
- Software reliability,
- Maintainability,
- Reliability prediction,
- Physics of failure.

The 1984 April special issue marking the IEEE Centennial is useful as a model.

Submit abstracts to the Guest Editor; state explicitly that they are for this special supplement. The papers will be refereed, and comments provided to authors.

Guest Editor

Anthony Coppola
201 Mill St.
Rome, New York 13440 USA

phone: [1] 315-339-7075
fax: [1] 315-337-9932
e-mail: acoppola@rome.iitri.com

Schedule

- Abstracts due . . . . . 1998 Jan 27
- Notification of abstract acceptance . 1998 Feb 14
- Draft papers due . . . . . 1998 Apr 20
- Final papers due . . . . . 1998 Aug 30

Special Feature

A special feature (similar to *Correspondence* items) will present interesting memories of those in the product assurance professions. Short reminiscences may be submitted up to 1998 August 10. Reminiscences will not be refereed, but will be edited to conform to our requirements and to fit the available space. ◄TR►