RC21619(96924)17MAY1999 Computer Science/Mathematics

IBM Research Report

FARACost: An Adaptation Cost Model Aware of Pending Constraints

Daniela Roşu

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

Karsten Schwan

Georgia Institute of Technology
Atlanta, Ga. 30332-0250

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

FARACost: An Adaptation Cost Model Aware of Pending Constraints*

Daniela Roşu
IBM Research Division
Thomas J. Watson Research Center
Yorktown Heights, NY 10598
drosu@watson.ibm.com

Karsten Schwan
Georgia Institute of Technology
Atlanta, Ga. 30332-0250
schwan@cc.gatech.edu

Abstract

The perturbations induced by adaptation and resource allocation decisions on the adapted applications may have the undesirable side effect of causing timing constraint failures. In order to benefit from available adaptation capabilities yet avoid critical timing failures, the dynamic resource allocation mechanism should be aware of the perturbation induced by its decisions. Therefore, the impact of adaptation on short-term performance should be considered a first-class decision criterion, along with traditional criteria such as long-term performance and application criticality. Towards this end, we propose the *FARACost*, an adaptation cost model that captures the impact of application-specific adaptation procedures and uses this information to evaluate adaptation choices. Experimental evaluations with two applications demonstrate that the use of models like *FARACost* reduces or prevents pending timing constraint failures, while leading to long-term performance improvements. The experiments are conducted in a cluster environment with a fully implemented infrastructure for adaptation and resource allocation based on the FARA framework.

*Funded in part by DARPA through the Honeywell Technology Center under Contract No. B09332478 and Contract No. B09333218, and by NSF equipment grants CDA-9501637, CDA-9422033 and ECS-9411846.

1 Introduction

Dynamic resource allocation is a critical component in any system expected to deliver predictable performance while subject to unexpected workloads. Sample systems range from embedded subsystems like communication infrastructures [11, 2, 22], to embedded systems composed of a moderate number of tasks running on homogeneous platforms, like radar[14] and robot control systems[23], and to systems composed of dynamic sets of components executing on heterogeneous distributed computing platforms, like the Navy Surface Combatant Ship SC-21[10].

In such systems, unexpected workloads are caused by new application arrivals and, more critically, by changes in the resource requirements of existing applications. Techniques based on resource reallocation and application-specific adaptation are typically used to solve the resulting overload situations. Previous research has proposed a multitude of algorithms and policies for negotiation among running applications [1, 6] or for adaptation of an application's resource and quality requirements [8, 7, 12]. In addition, our previous work[20] has identified and demonstrated that delays associated with adaptation and resource allocation (ARA) can impact on the short-term performance of applications with pending timing constraints.

This paper further reinforces our contention that ARA decisions should consider the short-term adaptation impact as a first-class criterion. Specifically, we develop *FARACost*, an adaptation cost model that captures the characteristics of adaptation procedures and allows us to use this information to estimate their impacts on pending timing constraints. Consequently, an adaptation decision method can estimate the short-term impact of each of the available adaptation options and use it along with decision criteria related to long-term performance.

Experimental evaluations with two representative applications demonstrate the benefit of this approach. Namely, we demonstrate that adaptation methods accounting only for long-term performance cannot control the timing failures that occur during adaptation enactment. We illustrate how *FARACost* enables us to compare the available adaptation choices and select the one with the best balance of long- and short-term performance. Experiments are conducted in a cluster environment with an ARA infrastructure based on the FARA framework[19, 21].

The applications considered as examples in this presentation are representative of military and commercial systems that integrate several independently authored adaptive applications [9, 10]. Namely, we consider a video transmission application, similar to the model in [6] and a graphics rendering application, similar to a real-time sensor processing application [14, 13, 7, 12].

Related Work. Previous solutions for adaptive resource allocation in real-time systems address both generic[1, 8] and specific application types. The set of specific types includes multimedia stream processing[6, 12] or pipelined-imprecise computations[7]. The characteristic common to these solutions is that they restrict their interest to maximizing long-term benefits. For instance, the most commonly considered benefit formulations include: (1) the number of critical applications being denied service [6], (2) the number of successfully serviced tasks[5, 18], and (3) the overall value of the system, where each job carries a benefit and a cost[1, 2, 7, 17] or where each system state carries a value[3].

Previous work has acknowledged the importance of considering the perturbation induced on target applications by the ARA decision. For instance, methods have been proposed to control decision overheads, as exemplified by load distribution methods[18] that consider the deadline of the real-time task as a bound for the distributed decision process. Similarly, the incremental decision algorithm proposed in [8] can trade decision quality for decision time. However, the decision process itself does not account for the potential perturbations resulting from the enactment of adaptation and reallocation decisions. Perhaps, this is because previous research has focused mainly on adapting applications upon arrival[7], and on adaptations involving relatively simple enactment procedures, like a rate adaptation which involves only parameter modification and timer resetting[12]. Finally, some studies that consider the allocation or the reallocation of currently active real-time tasks assume that the related procedures can be completed in the interval between two execution cycles[3], or they do not mention this issue at all[1, 2].

FARACost complements previous solutions by providing abstractions and methods that allow us to capture the characteristics of adaptation procedures and to quantify their impacts. By using *FARACost* previously proposed negotiation methods can be extended to control both the short- and the long-term impacts of adaptation.

Overview. Section 2 presents the underlying principles of *FARACost*. Section 3 presents the sample applications used in our experiments. Section 4 presents the implementation of *FARACost*, and Section 5 presents experimental results that illustrate the necessity and the benefits of considering the short-term impact of adaptation as a first-class decision criterion. Section 6 summarizes our contributions.

2 The *FARACost* Adaptation Cost Model

The principal contributions of this paper are (1) the identification of a set of application-specific attributes that enable the evaluation of short-term adaptation impacts and (2) the design of a method for evaluation of adaptation choices addressing both long- and short-term impacts. Our solutions are integrated in the *FARACost* adaptation cost model. *FARACost*'s abstractions and methods address a generic application model and systems with multiple, independently authored applications. Using *FARACost* a decision method can perform the tasks necessary for adaptation choice evaluation:

- determine the characteristics of adaptation procedures;
- estimate the impacts of enactment delays; and
- quantify short- and long-term impacts.

In the following, focusing on these tasks, we introduce the principles of the *FARACost* design, starting with a brief presentation of the FARA framework, the context in which *FARACost* is designed and implemented.

FARA. FARA is a framework for building ARA infrastructures that supports systems that include multiple resource providers and that run multiple concurrent, independently authored applications [9, 16]. FARA offers an integrated set of abstractions and mechanisms for building ARA infrastructures. FARA includes abstractions that (1) describe an application's adaptation capabilities and (2) provide information necessary for the control of short- and long-term impacts of adaptation decisions. The types of adaptations considered by FARA include changes of requirement levels, called 'level-adaptations', changes of the set of requirements, called 'set-adaptations', and changes of service location (i.e., migration). The following features of FARA application model are relevant for the design of *FARACost*:

• *Service requirements are distinguished from resource requirements.* Applications describe their requirements in terms of services, and service and resource providers interact in terms of resources.

• *Quality requirements and service requirements are captured by different abstractions.* This allows us to capture performance constraints that cannot be expressed by the parameters of a single service requirement, such as the end-to-end latency for a pipelined application. As part of a quality requirement specification, the application indicates the set of service requirements that can impact on its level, henceforth called *dependent requirements*.

Characterize Adaptation Procedures. *FARACost* includes abstractions that enable an application to characterize it adaptation procedures with respect to their *resource needs*, *start and execution delays*, and *intrinsic quality reductions*. This is motivated by the following insights.

Our experiments with adaptive real-time applications, such as parallel distributed C3I applications [20], indicate that the overhead of adaptation enactment involves operations at both application and system level, consuming resources like CPU and network bandwidth. Furthermore, some adaptation procedures may have resource requirements that exceed the corresponding requirements after the adaptation, in steady state. For instance, consider a vision application implemented using a data-decomposition approach and comprised of several concurrent and cooperating modules[7, 12]. In addition to the low-overhead rate and output quality adaptations[1, 7, 12], this application may adapt with respect to its concurrency level[13]. The enactment of this adaptation entails starting new threads and performing inter-thread communication, resulting in significant enactment overheads and delays. Similarly, adaptations that cause task migration[1] may incur large enactment overheads due to the cost of transferring application state[4]. In addition, the bandwidth that is required for state transfer might be larger than the bandwidth required for its regular event-processing procedures.

The delays and overheads associated with the adaptation procedure can be difficult to determine statically, as assumed possible in [15]. First, the completion times of adaptation procedures that entail task creation and migration are difficult to predict as they depend on current system load and on the criticality of ongoing activities. Second, the start time of an adaptation procedure may be delayed because the application is designed to adapt only at specific points during its computation.

Finally, quality reductions are not only due to delays in execution, but they might be caused also by the sequence of operations performed by the adaptation procedure. For instance, in the context of a video transmission application, frame loss results from an adaptation procedure that reduces the Sender's memory requirements by discarding a memory buffer segment that contains untransmitted video content.

Characterize Impact of Execution Delays. The execution delays caused by the adaptation procedure may impact the levels of quality requirements. Besides the length of the delay, this impact depends on the tolerance to delay of each of the affected quality requirements. For instance, for a sensor-processing application that can tolerate missing a sample, execution delays comparable to its sampling period do not have critical impacts. Therefor, *FARACost* allows an application to specify the range of acceptable delays for each of its quality requirements. By considering the actual number of timing failures rather than the execution delay itself, this method factors out the timing characteristics of the affected applications, thereby enabling a consistent comparison of solutions that affect multiple applications.

Evaluate Adaptation Choices. The goal of adaptation choice evaluation is to combine adaptation delays, delay tolerance, and long-term benefits into values useful for comparing the choices that are available to an ARA decision. The *FARACost*'s evaluation method considers the impact of adaptation as reflected by the following metrics [20]:

- short-term performance failures, accounting for timing failures and intrinsic quality reductions; and
- long-term performance laxity variation, accounting for the performance variation relative to the minimal acceptable performance.

For each adaptation option, this method computes a cost-performance factor, called the *overall impact coefficient*. The 'cost' represents the short-term performance failures, and the 'performance' represents the long-term performance laxity variation. As adaptation choices may affect multiple quality requirements and applications, the 'cost' and 'performance' factors are based on a *value model*, which translates to corresponding metrics in values that reflect the relative importance of quality requirements. This model associates larger delay penalties and larger improvement benefits with the more critical requirements.

3 Sample Applications

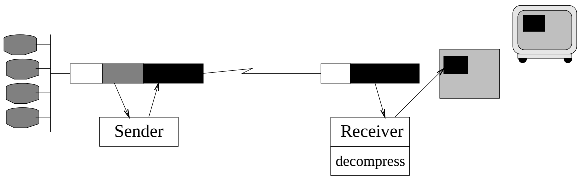


Figure 1: Video Transmission Application.

Video Transmission Application. The video transmission application considered in our study is similar to the model in [6] except that it has more adaptation capabilities. The implementation is based on the MPEG Video Software Decoder from U.C. Berkeley. The application has two modules, the Sender and the Receiver (see Figure 1). The Sender implements a two-buffer-based scheme for reading a stream from disk and sending it, over the network, to the Receiver. The Receiver decompresses the data and stores it in the display buffer. The two modules are periodic tasks, with their execution driven by periodic timers. The performance requirements of this application concern the frame rate (reflected by the inter-frame interval), the stream resolution, and the display color. The hosts on which the two modules should run are fixed. Tables 1-2 list the service and quality requirements of this application. According to user preferences, the color requirement is the most important, followed by resolution and frame interval.

The application is capable of the following level-adaptations: (1) vary the frame rate or resolution requirements, thereby changing the computation, communication, and memory

Table 1: Video Transmission Application: Service Requirements.

Id	Type	Placement
Steady State Procedures		
cpu1	PERIODIC EXECUTION	Sender
dataBuffs	PERMANENT MEMORY	Sender
comm12	REAL TIME CHANNEL	(Sender, Receiver)
cpu2	PERIODIC EXECUTION	Receiver
inBuff	PERMANENT MEMORY	Receiver
displayBuff	PERMANENT MEMORY	Receiver
window	X11	Receiver
Adaptation Procedures		
colorCpu	TIMESHARING EXECUTION	Receiver

Table 2: Video Transmission Application: Quality Requirements.

Identifier	Importance	Delay Tolerance
frame interval	4	(cpu1.Period,1), (cpu2.Period,5)
resolution	10	none
color	15	none

Table 3: Video Transmission Application: Adaptation Cost Attributes.

Adaptation Procedure	Delay Paths	Start Delay
color change	{ colorCpu }	cpu2.Period

requirements of the two modules; (2) vary the display from color to monochrome, thereby lowering the Receiver's computation requirements; and (3) vary the period of the Sender's execution, thereby lowering memory requirements on the Sender and increasing the period of its CPU, disk, and bandwidth reservations.

Table 3 presents the attributes of the related adaptation procedures. Except for the color-related adaptation, the resource needs of these adaptations are almost insignificant. However, their completion requires up to two Sender periods, the first because the adaptation procedure can only start at the beginning of an execution slice, and the second because of the need to resize the I/O buffers gradually such that no content is lost. The color-related adaptation requires a small amount of computation to initialize the decompression context.

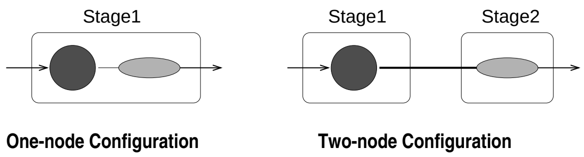


Figure 2: Graphics Rendering Application.

Graphics Rendering Application. The second adaptive application is a graphics rendering application whose computation pattern and adaptation capabilities make it similar to a sensor processing application. This application uses OpenGL primitives to periodically render two spinning objects. The types of the rendered objects can vary, and along with them, the output quality and level of service requirements. Better output quality results in higher computation requirements. The application can execute in one- and two-node configurations, and it is flexible about how much computation is performed at each site and about where its modules execute (see Figure 2).

Table 4: Graphics Rendering Application: Service Requirements.

Id	Type	Placement
Steady State Procedures		
cpu1	PERIODIC EXECUTION	Stage1
buff1	PERMANENT MEMORY	Stage1
renderer1	OGL	Stage1
comm12	REAL TIME CHANNEL	(Stage1, Stage2)
cpu2	PERIODIC EXECUTION	Stage2
buff2	PERMANENT MEMORY	Stage2
renderer2	OGL	Stage2
Adaptation Procedures		
tsCpu1	TIMESHARING EXECUTION	Stage1
tsCpu2	TIMESHARING EXECUTION	Stage2
rlcomm12	RELIABLE CHANNEL	(Stage1, Stage2)
remote	START REMOTE PROCESS	Stage1

Table 5: Graphics Rendering Application: Quality Requirements.

Identifier	Importance	Delay Tolerance
deadline loss	5	(cpu1.Period,8)
outcome	10	none

Table 6: Graphics Rendering Application: Adaptation Cost Attributes.

Adaptation Procedure	Delay Paths	Start Delay
1-node to 2-node	{remote, tsCpu1, rlcomm12, tsCpu2}	cpu1.Period
2-node to 1-node	{tsCpu1}	cpu1.Period
level adaptations	{ tsCpu1}, { tsCpu2 }	cpu1.Period or cpu2.Period

Tables 4-5 list the service and quality requirements that characterize this application. The 'outcome' quality requirement corresponds to the output quality. The 'deadline loss' quality requirement corresponds to the constraint that the first component, *Stage1*, should periodically "read the sensor", not missing more than a pre-defined number consecutive deadlines.

The configuration changes affect the amount of computation performed at each node. The one- to two-node configuration change lowers the computation load on Stage1's node and the reverse change eliminates the load on Stage2's node. Besides these set-adaptations, the application is capable of the following level-adaptations: (1) shift work from one stage to another while maintaining the same quality level, and (2) reduce the amount of work performed by one or both stages at the expense of reducing the outcome level. These adaptations affect the computation requirements of one or both components, and, possibly, their communication requirements as the amount of data sent by Stage1 is inversely related to the amount of its computation requirements.

Table 6 presents the attributes of the related adaptation procedures. In all cases, the adaptation procedure cannot be performed while an event is being processed. Consequently, its initiation is delayed until the beginning of the next execution slot of the adapted component. The procedure for the one- to two-node configuration change requires the initialization of a new thread on the remote node, the transmission of the current execution context, and the initialization of the rendering process. The procedures related to level-adaptations only initialize the rendering of the new object(s), using an amount of computation resources which increases with the size (and the value) of the initialized object.

4 *FARACost* Implementation

The *FARACost*'s implementation has four components: (1) *the adaptation impact model*, which captures information about the adaptation procedure overheads and service requirements; (2) *the delay tolerance model*, which captures information about the tolerance of quality requirements to execution delays; (3) *the value model*, which translates quality levels in values that can be consistently compared across requirements and applications; and (4) *the adaptation choice evaluation method*, which quantifies the impact of an adaptation choice.

The Adaptation Impact Model. The adaptation impact model is captured by the *ARAImpact* abstraction. With each adaptation procedure may be associated an *ARAImpact* entity. The *service requirements* are specified explicitly. The *start delay* can be expressed directly, by a value, or indirectly, by a parameter of a service requirement in the before-the-adaptation set. For instance, the video application uses the indirect approach, specifying the start delay by its worst-case estimate, which is the period of the Sender's execution. *Execution delay*, can be specified by an explicit value, or implicitly, by *delay paths*. A delay path is an ordered subset of the *ARAImpact*'s service requirements, and its corresponding delay is the sum of their (estimated) completion times. With an adaptation procedure can be associated several delay paths, the overall execution delay being the maximum of the per-path delays. For instance, in the context of the graphics application, the execution delay for the level-adaptations affecting the two application components is described by two delay paths, each with one requirement (see Tabel 6). The delay paths are evaluated by the ARA infrastructure in cooperation with the service providers or based on the service models integrated in the infrastructure. In contrast to explicit values, the use of delay paths allows an application to provide accurate information independently of the characteristics of the system on which it runs. The *intrinsic quality reductions* are specified explicitly, indicating the quality requirement and the level reduction that occurs during the execution of the adaptation procedure. None of our sample applications exhibits such reductions.

Delay Tolerance Model. With each quality requirement can be associated a *delay tolerance* abstraction. This abstraction specifies (1) the *delay unit*, which represents the maximum time interval that can elapse between two consecutive executions of any of its dependent service requirements before a failure is declared, and (2) the *tolerance coefficient*, which represents the ratio of the delay unit into the maximum acceptable delay. The delay unit can be specified explicitly or it can be associated with a service requirement parameter. For instance, in our experiments, the graphics application tolerates up to eight consecutive misses of the Stage1's deadline; therefore, the delay tolerance of the 'deadline loss' requirement is defined by a delay unit equal to the period of Stage1 execution and the tolerance coefficient is eight. Multiple delay tolerance entities can be associated with a quality requirement, like for the video application's 'frame interval', which is sensitive to delays in execution of both Stage1 and Stage2.

Value Model. *FARACost*'s value model reflects both the potential interest in increasing a quality level and the penalty of lowering it. Because of our interest in highlighting the negative adaptation impacts, this model gives more weight to (1) performance reductions vs. performance improvements, (2) performance below acceptable levels vs. performance within acceptable levels, and to (3) variations of more important vs. less important requirements. Towards this end, we use the following value function (μ), called *impact function*, where α represents the importance coefficient and β the level relative to minimum acceptable:

$$\mu(\alpha, \beta) = \begin{cases} \log \alpha \log (1 - \beta), \text{ when } \beta < 0, \text{ and} \\ (1+\beta)^{-\frac{1}{\alpha}} \quad\quad\quad , \text{ when } \beta >= 0 \end{cases}$$

Figure 3 illustrates the variation of the impact function for two importance coefficients, demonstrating that this selection satisfies our design goals.

With each quality requirement are associated two importance coefficients – one for the long-term performance and one for short-term performance. Both coefficients are larger than one, and they may be equal, like for our sample applications. The long-term coefficient reflects the relative *benefit* of long-term performance above minimum acceptable level, and short-term coefficient reflects the relative *penalty* of short-term performance below acceptable levels.

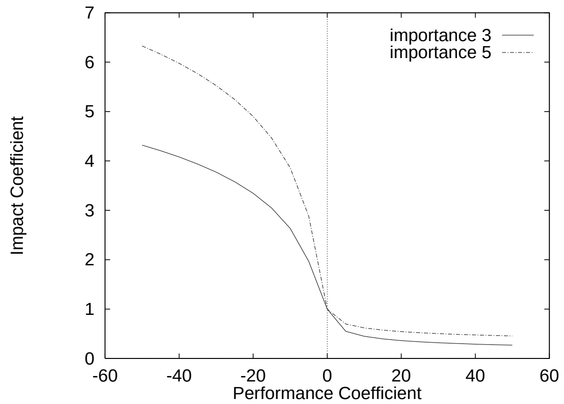


Figure 3: Impact function (β = performance coefficient, α =importance)

$$\begin{aligned} P_S(D,q) &= \min\Big\{AcceptableLoss(q) - ExpectedLoss(D,q), \\ &\quad \frac{IntrinsicReduction(D,q)}{MinimLevel(q)} \cdot 100\Big\} && (1) \\ P_L(D,q) &= \frac{ExpectedLevel(D,q) - CurrentLevel(q)}{MinimLevel(q)} \cdot 100 && (2) \end{aligned}$$

$$\begin{aligned} I_S(D,q) &= \mu(Imp_S(q), P_S(D,q)) \quad I_L(D,q) = \mu(Imp_L(q), P_L(D,q)) && (3) \\ I_D(D) &= \frac{ExpectedCompletion(D)}{RequestDeadline(D)} && (4) \end{aligned}$$

Table 7: Notation for Decision Evaluation.

q	quality requirement
D	ARA choice
$Imp_S(q)$	short-term importance coeff. of q
$Imp_L(q)$	long-term importance coeff. of q
$P_S(D,q)$	short-term performance coeff.
$P_L(D,q)$	long-term performance coeff.
$I_S(D,q)$	short-term performance impact coeff.
$I_L(D,q)$	long-term performance impact coeff.
$I_D(D)$	deadline impact coeff.
$O(D,q)$	requirement-level overall impact
$\mathcal{O}_S(D)$	short-term system-level impact
$\mathcal{O}_L(D)$	long-term system-level impact
$\mathcal{O}(D)$	system-level impact
$\mathcal{F}_S$	short-term aggregate function
$\mathcal{F}_L$	long-term aggregate function

$$\begin{aligned}
\mathcal{O}_L(D) &= \mathcal{F}_L\left\{I_S(D,q), q \text{ quality requirement}\right\} &(5)\\
\mathcal{O}_S(D) &= \mathcal{F}_S\left\{I_D(D), I_S(D,q), q \text{ quality requirement}\right\} &(6)\\
\mathcal{O}(D) &= \mathcal{O}_S(D) \cdot \mathcal{O}_L(D) &(7)\\
D_1 \text{ better than } D_2 &\iff \mathcal{O}(D_1) < \mathcal{O}(D_2)
\end{aligned}$$

Adaptation Choice Evaluation Method. The *FARACost* method for adaptation choice evaluation consists of the following steps (see the notation in Table 7): *1.* Compute *short and long-term performance coefficients* for each quality requirement affected by the adaptation (Equation 1 and 2, respectively). Lower performance coefficients reflect lower expected performance. For long-term, positive performance coefficients correspond to performance improvements. For short-term, positive performance coefficients correspond to performance reductions within acceptable limits.

2. Compute *short- and long-term quality impact coefficients*, applying the value model to the corresponding performance coefficients (Equation 3);

3. Compute the *deadline impact coefficient* (Equation 4);

4. Compute the *overall impact coefficient(s)* by aggregating per-state impact coefficients at component, client and/or system levels (Equations 5-7). Finally, when comparing two adaptation choices, the lower the overall impact coefficient, the better is the choice.

ExpectedCompletion is computed as the sum of start and execution delays specified by the *ARAImpact*. If multiple adaptation procedures are involved, the *ExpectedCompletion* is the maximum value across the adaptations that results in the release of to-be-reallocated resources. *ExpectedLoss* is computed based on the execution delays and delay tolerances specified by *ARAImpact*. If a quality requirement has multiple delay tolerance parameters, the pair with the maximum *ExpectedLoss* is considered. *IntrinsicReduction* is the reduction specified by *ARAImpact*, if any. Sample aggregation functions, $\mathcal{F}_L$ and $\mathcal{F}_S$, are sum, maximum, and minimum. For instance, maximum or sum can be used for aggregation of short-term impact in a system in which timing constraint failures are critical. Minimum-based aggregation can be used for long-term impact in a best-effort system not concerned with providing fair service.

This method evaluates the incremental change as it considers only the quality requirements affected by the adaptation. In order to evaluate the entire system state, the evaluation steps are applied to all quality requirements in the system and the long-term performance coefficient ($P_T(D, q)$) reflects the variation with respect to the minimum acceptable level.

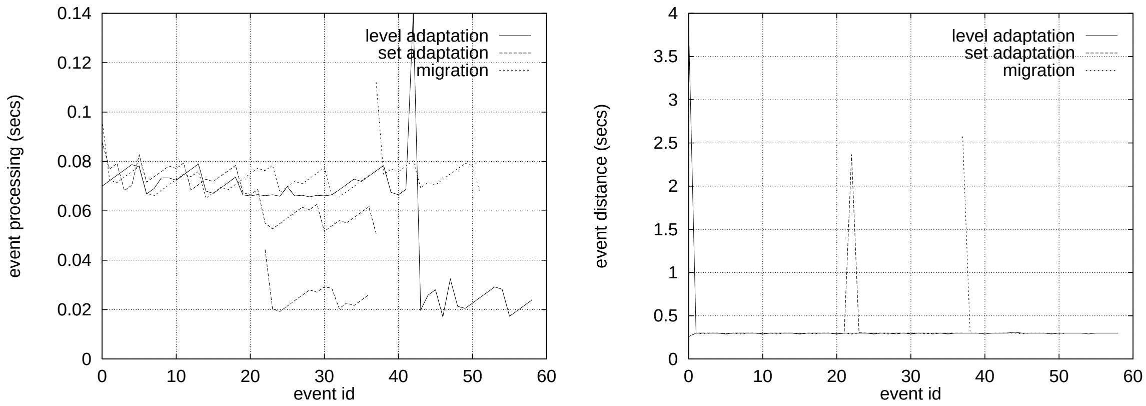
Using *FARACost*. In its interaction with the ARA infrastructure, an application uses the *ARAImpact*, delay tolerance, value abstractions to specify its characteristics along with its start-time resource allocation request. Depending on the infrastructure, some of these characteristics can be updated at runtime. FARA enables runtime updates of the *ARAImpact* and importance coefficients.

5 Experimental Evaluation of *FARACost*

In this section, we demonstrate that adaptation cost information is necessary for making adaptation decisions about applications that have multiple adaptation options and operate in systems with dynamic sets of applications. First, we demonstrate that in the context of a single application, various adaptation choices can have significantly different impacts on short-term performance. Namely, in the context of the graphics rendering application, we show that for three adaptations that reduce the load on Stage1's node, the number of consecutive timing failures varies from zero to eight. Second, we demonstrate that long-term-driven adaptation methods cannot control the timing failures that occur during adaptation enactment. Specifically, we experimentally compare an importance-driven method [9, 1] and a cost-driven method, showing that the former cannot shield a higher-importance application from the adaptation delays of a lower-importance application. Finally, we demonstrate that the *FARACost*-based evaluation of adaptation choices enables us to select the choice with the least (expected) impact on the currently pending timing constraints. This shows that an abstraction like *FARACost* is a necessity in systems in which the set of pending timing constraints may vary in time.

These experiments are run in a cluster of Sun Ultra 30 workstations in which a prototype FARA-based ARA infrastructure provides for centralized ARA decisions; the decision core runs on a dedicated machine. Applications connect directly to the infrastructure, issuing resource allocation requests at start time. The infrastructure calls back the application to request the complete specification of level-adaptations and to inform about adaptation decisions.

Adaptation Impact Varies with the Adaptation Choice. This experiment starts with the graphics application running in one-node configuration. An adaptation is triggered by the arrival at the same node on which the graphics application is running of a new periodic application (a micro-benchmark) with fixed placement constraints. The computation requirements of this application cannot fit in the available resources on the node. In this context, the graphics application offers three adaptation choices – migration, level-adaptation, and configuration change (i.e., set-adaptation). Each of these choices may be selected for enactment by some negotiation decision as each of them minimizes some interesting long and short-term performance criteria. Namely, the migration choice, moving the entire computation to another node, does not affect the graphics application's long-term performance. The level-adaptation, reducing the computation requirements to the expense of a reduction of the 'outcome' level, minimizes the start-time delay of the newly arrived application. Finally, the set-adaptation, switching the application to a two-node configuration, balances the quality degradation of the graphics application and the start-time delay of the newly arrived application.



(a) (b)

Figure 4: Adaptation Choices of the Graphics Rendering Application. (a) Change in computation requirements; (b) Delays in the periodic execution.

Figure 4 presents the impacts of the three adaptation choices on long-term computation requirements (Plot (a)) and on short-term performance as reflected by the delays in periodic execution (Plot (b)). Plot (a) presents the computation requirements for all of the nodes of the adapted application. For instance, for the set-adaptation, a new node is created at event 22. Plot (b) illustrates the delays in periodic execution as reflected by the variation of the event inter-arrival time (i.e., event distance). We note that the level-adaptation has almost no impact. This is because its enactment overhead is small in comparison to the execution period. In addition, the delay incurred with the migration option is larger than the tolerance to delay of this application (see Table 5). The large impacts of migration and set-adaptation are due to the costs of task migration, which in our implementation involves an 'rsh' command (see 'START REMOTE PROCESS' requirement in Table 4). Overall, this experiment illustrates that decision criteria that lead to acceptable long-term performance can result in different impacts on pending timing constraints. Therefore, when making the decision it is necessary to evaluate these impacts using models like *FARACost*.

Optimal Decisions Combine Importance and Adaptation Delay Attributes. We experiment with two decision methods. Namely, focusing only on short term, the cost-driven method selects the adaptation with the least execution delay irrespective of the importance of the affected application. In contrast, focusing only on long term, the importance-driven method takes a first-fit approach, considering the applications in non-decreasing order of their importance coefficients[9, 6].

The experiment involves two periodic microbenchmarks with fixed placement constraints and able to trade computation for memory requirements. The two applications have different importance coefficients and adaptation overheads. Namely, when switching to a configuration with lower memory requirements, the low-importance application executes a number of floating-point operations, incurring an overhead of $7.5\mu secs$ for each 100 loops. The application specifies this overhead using the *execution delay* parameter of the associated *ARAImpact* entity. For a similar adaptation, the high-importance application incurs no adaptation overhead.

Adaptation is triggered by the arrival of the high-importance application to the node where the low-importance application is running. The overloaded resource is the memory. The high-importance application has a $50ms$ start-time constraint. The importance-driven method selects the adaptation choice offered by the low-importance application, while the cost-driven application selects the adaptation choice offered by the high-importance application.

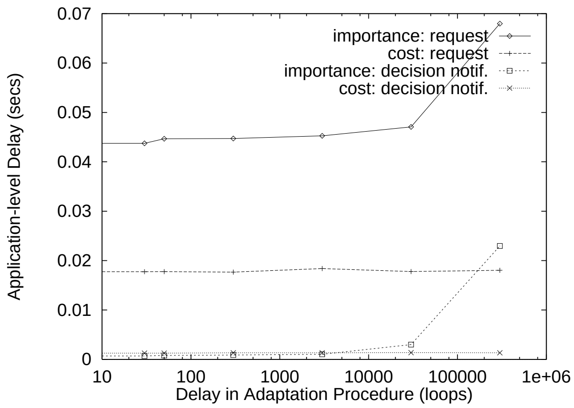


Figure 5: Delays with adaptation cost and importance-driven adaptation methods.

Figure 5 presents the delays observed at application level when the number of loops performed in the adaptation procedure by the low-importance application varies. Namely, the plot presents (1) the overhead of handling decision notification (which includes the adaptation procedure) and (2) the request response time, which represents the latency of allocation request observed at user-level.

Comparing the request response time with the start-time constraint, we notice that the optimal solution depends on the adaptation overhead of the low-importance application. The importance-driven decision results in timing failures when this overhead is very large. In contrast, the cost-driven decision prevents timing failures but results in undue value reductions when the adaptation overhead is low. It worth to be mentioned that the differences between the request time and the notification overheads result from the interactions between applications and the ARA infrastructure (the two methods incur almost the same decision overhead – about $1ms$).

This experiment illustrates that in the context of real-time applications, optimal decisions need to combine long- and short-term performance criteria, and to compare the pending timing constraints with the expected adaptation overheads as considered by *FARACost*.

Addressing the Pending Timing Constraints. In this experiment, the negotiation method uses *FARACost* to determine the adaptation choice with minimal short-term impact and maximal long-term performance. Figure 6 presents this negotiation method. The impacts on long and short-term performance are computed using the model in Section 2. The available adaptation choices are evaluated and ranked according to their short and long-term impacts. The per-state aggregation method is one plus the sum over all affected quality requirements of the impact coefficient values in excess of 1. Therefore, when all choices have acceptable short-term impacts, the selection is determined by the long-term value, just like in the negotiation methods proposed by related research. The identification of the available adaptation choices is based on the description of an application's adaptation capabilities enabled by the FARA Adaptation Model[21]. This method can only result in adaptations that affect a single application.

Table 8: Impact attributes for the available adaptation choices.

Choice	Start (ms)	Execution (ms)	Completion (ms)	Short-term			Long-term	
				Perf. Coeff.	Impact		Perf. Coeff.	Impact
					Aggregate	Actual		
0	20.76	1944.38	1965.14	-33.86	5.71	5.71	0	1.00
1	20.76	1.12	21.88	4.97	1.00	0.69	-50	9.05
2	2.37	0.05	2.42	8.00	1.00	0.57	-80	11.90
3	5.49	160.00	165.49	6.33	1.00	0.61	-80	6.09

Table 9: Adaptation decision for various request deadlines.

Deadline (ms)	Selection	Decision (overall)	Negotiation Decision	Allocation Decision	User Interaction
1000	Choice 3	10.102	3.993	2.190	3.352
100	Choice 1	8.984	3.524	1.605	3.188
10	Choice 2	9.567	3.849	1.965	3.236

The experiment involves the graphics rendering application, the video application, and a periodic application with fixed placement constraints and start-time constraints. The graphics application executes in two-node configuration, with Stage1 on the same host on which the video Receiver is running. The decision is triggered by the arrival of the periodic application at the host where Stage1 and Receiver are running.

In this context, the negotiation method has four choices, concerning both the graphics and the video applications: [**Choice 0**] Migrate Stage1 to another node. [**Choice 1**] Adapt the graphics application such to reduce the level of the 'outcome' requirement and the load induced by Stage1. [**Choice 2**] Adapt the video application by switching from color to monochrome display, and thereby reducing the level of the 'color' requirement. [**Choice 3**] Adapt the video application by reducing the display rate, and thereby reducing the level of the 'frame interval' requirement.

For each of these choices, Table 8 presents the expected delays, and the impact on quality requirements. Note that besides the application-specific attributes (see Tables 2,3,5,6), the adaptation method uses the cost characteristics of the migration operation provided by an infrastructure-level model.

Table 8 shows that the aggregate short-term impact is zero for all choices except *Choice 0* (migration), because their expected delays are smaller than the maximum delay tolerated by the related quality requirements. Table 9 presents the selection made for each of the start-time constraints considered in our experiments. These selections minimize long and short-term impact coefficients. Thereby, the selected adaptation is not always the one with the best long-term quality levels, but the pending timing constraints of all active and just

Input: Client C_f, failed service requirement S_f and overloaded resource r_f
Output: Client C_a and adaptation option $A \in C_a.Adaptations$, or **NULL**

```
  𝒜 = {A ∈ C_f.Adaptations | A lowers S_f load on r_f}
 for each client C using r_f
   Add to 𝒜 all adaptation options that lower the load on r_f
 for each A ∈ 𝒜
   𝒪_S(A) = 𝒪_L(A) = 0
   compute I_D(A)
   if (I_D(A) > 1)
     remove A from 𝒜; continue
   𝒪_L(A)+ = max{0, I_D(A) − 1}
   for each q affected by A
     compute I_S(A, q), I_L(A, q)
     𝒪_L(A)+ = I_L(A, q)
     𝒪_S(A)+ = max{0, I_S(A, q) − 1}
   𝒪_S(A)+ = 1
  sort 𝒜 in non-decreasing order of 𝒪_S, 𝒪_L
  for each A ∈ 𝒜
   if ( Allocate(A) == SUCCESS) return (A and client)
  return NULL
```

Figure 6: Adaptation Cost-Driven Negotiation.

arrived applications are not at all or only lightly affected by the adaptation.

Table 9 also presents the various components of decision overhead, including negotiation, allocation, and user interaction. This data shows that the overall decision is made in about 10 ms(the worst case, being incurred for the decision that has to assign the largest number of requirements). The negotiation alone takes about 4 ms. Notice that these overheads are larger than the tightest start-time constraint considered in this experiment (10ms). This shows that in the estimation of completion times we have to consider also the decision overheads.

Overall, this experiment demonstrates that the ability to evaluate the adaptation delays and estimate their impacts on timing constraints allows the adaptation decision method to successfully solve requests with very different pending timing constraints and incur only minimal short-term performance penalty.

6 Conclusions

This paper proposes *FARACost*, an adaptation cost model that enables an ARA decision method to control its impact on both short and long-term system performance. The model the impact of adaptation on pending timing constraints a first-class criterion in an ARA decision, along with long-term performance attributes such as application importance and criticality. As also stated by our previous work[20], this is a necessity in systems with dynamic requirements and multiple adaptive applications. Experimental results, produced with a

FARA-based ARA infrastructure and sample applications, demonstrate that the information captured by *FARACost* is necessary for evaluation of adaptation impacts and that the control of short-term impacts is feasible and benefits the adapted applications. In the future, we consider the design of negotiation methods that can produce solutions involving multiple applications and further refinements of *FARACost* towards a more accurate evaluation such adaptations.

References

[1] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control. *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, June 1997.

[2] T. F. Abdelzaher and K. G. Shin. End-Host Architecture for QoS-Adaptive Communication. *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, June 1998.

[3] T. Bihari and K. Schwan. A Comparison of Four Adaptation Algorithms for Increasing the Reliability of Real-Time Software. *Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 1988.

[4] J. Casas, D. Clark, P. Galbiati, R. Konuru, S. Otto, R. Prouty, and J. Walpole. Adaptive load distribution systems for PVM. *Proceedings of the Supercomputing*, pages 390–399, Nov. 1994.

[5] Y.-C. Chang and K. G. Shin. Optimal Load Sharing in Distributed RT Systems. *Journal of Parallel and Distributed Computing*, pages 38–50, Sept. 1993.

[6] J. Huang, Y. Wang, and F. Cao. On Developing Distributed Middleware Services for QoS- and Criticality-based Resource Negotiation and Adaptation. *Honeywell Technical Report SST-P97-012*, June 1997.

[7] D. Hull, A. Shankar, K. Nahrstedt, and J. W. S. Liu. An end-to-end QoS model and management architecture. *Proceedings of the IEEE Workshop on Middleware for Distributed Real-time Systems and Services*, pages 82–89, Dec.,1997.

[8] J. Jehuda and A. Israeli. Automated Meta-Control for Adaptable Real-Time Software. *Real-Time Systems Journal*, 14(2):107–134, Mar. 1998.

[9] M. B. Jones, P. J. Leach, R. Draves, and J. Barrera III. Modular Real-Time Resource Management in the Rialto Operating System. *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 12–17, May 1995.

[10] G. Koob. Quorum Program Overview. *http:// www.darpa.mil/ito/ research/pdf_files/quorum_approved.pdf*, July 1998.

[11] R. Kravets, K. Calvert, and K. Schwan. Payoff Adaptation of Communication for Distributed Interactive Applications. *Journal for High Speed Networks Special Issue on Multimedia Networking*, 7(3/4), 1998.

[12] B. Li and K. Nahrstedt. A Control Theoretic Model for Quality of Service Adaptations. *Proceedings of the IEEE International Workshop on Quality of Service*, May 1998.

[13] R. C. Metzger, B. VanVoorst, L. S. Pires, R. Jha, W. Au, M. Amin, D. A. Castanon, and V. Kumar. C3I Parallel Benchmark Suite - Introduction and Preliminary Results. *Proceedings of the Supercomputing*, Nov. 1996.

[14] J. Molini, S. Maimon, and P. Watson. Real-Time System Scenarios. *Proceedings of the IEEE Real-Time Systems Symposium*, Oct. 1991.

[15] D. M. Nicol and P. F. J. Reynolds. Optimal Dynamic Remapping of Data Parallel Computations. *IEEE Transactions on Computers*, pages 206–219, Feb. 1990.

[16] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource Kernels: A Resource-Centric Approach to Real-Time Systems. *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, Jan. 1998.

[17] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Practical Solutions for QoS-based Resource Allocation Problems. *Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 1998.

[18] K. Ramamritham and J. A. Stankovic. Dynamic Task Scheduling in Hard Real-Time Distributed Systems. *IEEE Software*, 1(3):65–75, July 1984.

[19] D. Rosu, K. Schwan, and S. Yalamanchili. FARA – A Framework for Adaptive Resource Allocation in Complex Real-Time Systems. *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 79–84, 1998.

[20] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On Adaptive Resource Allocation for Complex Real-Time Applications. *Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 1997.

[21] D. I. Rosu and K. Schwan. Dynamic Resource Allocation for Adaptive Real-Time Applications. *Georgia Institute of Technology*, Oct. 1999.

[22] A. Sahoo, B. Devalla, Y. Guan, R. Bettati, and W. Zhao. Adaptive Connection Management for Mission Critical Applications over ATM Networks. *NAECON'98*, 1998.

[23] K. Schwan, T. Bihari, B. W. Weide, and G. Taulbee. High-Performance Operating System Primitives for Robotics and Real-Time Control Systems. *ACM Transactions on Computer Systems*, 5(3):189–231, Aug. 1987.