Adaptive Multi-Sensor, Multi-Actuator Media Workflow System for Interactive Arts*

Lina Peng
Computer Science and Engineering Dept.
Arizona State University
Tempe, AZ, 85287, USA.
lina.peng@asu.edu

K. Selçuk Candan
Computer Science and Engineering Dept.
Arizona State University
Tempe, AZ, 85287, USA.
candan@asu.edu

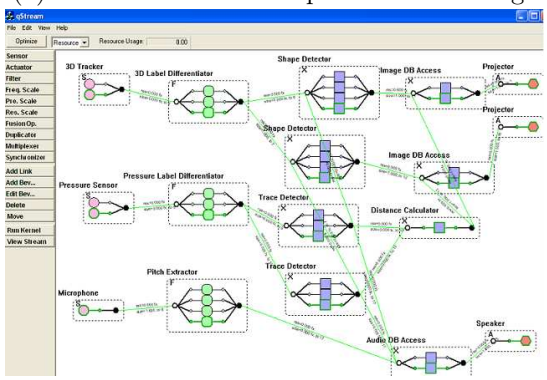
Abstract

We are presenting the Architecture for Interactive Arts (ARIA) middleware architecture for sensory/reactive environments, such as interactive performances. ARIA processes, filters, and fuses sensory inputs and actuates responses in real-time while providing various service guarantees. ARIA is deployed on a centralized stream processor that monitors and queries over distributed media sensors and outputs of the media workflows are sent to actuators with video and audio presentation capabilities. Due to rapid updates and the continuous nature of sensory data streams, approximation and adaptation techniques are necessary to optimize resource usage. We propose a mechanism that adapts media workflows by adjusting parameters of filters and fusion operators that dynamically control the speed of data streams and the sampling rate. In [1], we developed the ARIA middleware model, including the operators, performance models, and the optimization techniques for chain structured (single-sensor, single-actuator) media workflows. In this paper, we focus on development of flow optimization techniques in adaptive multi-sensor, multi-actuator workflow system. We experimentally evaluate the efficiency and the effectiveness of the algorithms.

1. Introduction

In [1, 2], we introduced the Architecture for Interactive Arts (ARIA) which is a sensor-to-actuator middleware architecture for sensory/reactive environments. ARIA processes, filters, and fuses sensory inputs and actuates responses in real-time while providing certain service guarantees. The Intelligent Stage at ASU is equipped with various sensors including: floor mat sensors to generate data for object localization and motion classification, microphone arrays for sound localization and beamforming, Vicon 8i Realtime system with 8 cameras for tracking the position and motion of human body that is appropriately marked with reflective markers, ultrasound sensors for object tracking, and video sensors for determining the presence or absence of marked persons, their relative spatial positions, and detecting certain simple events. Interactive

(a) An adult and a child performer on iStage



(b) *qStream* Workflow representation of the example scenario

Figure 1. An example interactive performance

* This work has been supported by NSF grant # IIS-0308268, "ARIA: Quality-Adaptive Media-Flow Architectures to Support Sensor Data Management".

performances that use this stage require an information architecture that captures, and processes various types of audio, video, and motion data while providing appropriate service guarantees. ARIA is designed for the purpose to be deployed in sensory/reactive environments. The following example scenario illustrates the tasks ARIA is being designed to accommodate:

- *Interactive Movement Poses.* Two performers, an adult and a child, are tracked by 3D-motion tracking devices and their locations on the stage are tracked using pressure sensors (Figure 1). ARIA continuously monitors the position of the body markers of the performers in 3D space, their positions and details of their gestures. The output of the 3D motion tracking is filtered through ARIA to obtain the shape and degree-of-confidence of the pose. The shapes are recognized by ARIA with the degree-of-confidence and the recognized shape drives the image from media repository to be projected on the stage. The degree-of-confidence of recognition and relative spatial locality of the performers are fused to generate the color pattern of images to be projected. The image and color information is transferred to projection actuators.

Currently, artists and choreographers are using best-efforts tools, such as Max/MSP and Pure Data (Pd) to develop such scenarios. Max/MSP [3] is an object-oriented graphical programming environment for music and multimedia in the MIDI standard format. Programs or patches are created by connecting objects together with patch cords. Events, which could be anything from a MIDI note to a mouse click, flow through the patch cords from one object to another. Pd is a real-time graphical programming environment for audio and graphical processing [4]. MAX and Pd provide a visual language framework to describe the connectivity amongst sensors, media filters, media fusion operators and actuators. However, both tools are very limited in their expressive power. Importantly, they lack the capability of describing realtime application constraints and operator properties. Consequently, they cannot take advantage of alternative media-flow strategies to provide realtime and Quality of Service (QoS) guarantees. In effect, they are best-effort solutions, where the use is limited to very specific hard-coded scenarios. In contrast, ARIA middleware architecture provides an *adaptive* flow management kernel, *qStream* , consisting of various adaptive filters and fusion operators to ensure service guarantees. The adaptive nature of *qStream* is due to the components of the data flow architecture that are programmable and adaptable: delay and quality characteristics of individual operators can be controlled via a number of parameter values. Properties of each object streamed between components are therefore subject to changes based on these parameters. The quality of an output object and the corresponding delay and resource usage depend on parameters assigned to the operators in the object's filter and fusion path.

1.1. Related Work

In media networking and application management domain, it has long been recognized that transmission of multimedia objects over networks must satisfy a set of quality of service (QoS) requirements. In addition, QoS-aware middleware, such as QOS Broker [15] and DQM [16], provides efficient QoS architectures for network resources. Q-RAM provides a set of QoS optimization schemes with discrete QoS options in the context of video-conferencing [11, 12]. Unlike ARIA which considers workflows, Q-RAM considers tasks as independent applications and imposes the quantification of the system utility on users via weighting the linear combination of all quality dimensions of all applications. [13] alleviates the requirement of explicit quantification by relying on a tunability interface. Aurora [18] focuses on QoS- and memory-aware operator scheduling and load shedding for coping with transient spikes in data streams.

In a media service composition framework [5, 6, 7, 8, 9, 10], the task is usually to communicate a media object or a stream from a source server to a media consumer, while the overlay routing nodes provide mostly application level services, such as transcoding and mixing. Recently, there has been a number of efforts in automatic composition of such distributed multimedia services. These include, SpiderNet [5], SAHARA [6, 7], SPY-Net [8], CANS [9], and Infopipes [10]. The focus of most of these works is to optimize the network resources and to prevent congestion. Most relevantly, SpiderNet [5] models a composite service using a directed acyclic graph with adaptive multimedia service components and aims finding mappings that satisfy the users' multi-constrained QoS requirements. In fact, it has been shown that, generally, this class of problems are NP-complete. We note that these works , [5] and others generally ignore the interdependencies between parameters where, for instance, resource consumption at a given node may depend on the size and precision of its inputs.

Various forms of multi-constrained optimal routing (MCOR) problems have been studied in the literature [19, 20, 21, 22] and it has been shown that generally this class of problems are NP-complete. In this paper we *do not* address explicit multi-criteria opti-

mization problems where multiple QoS criteria are being optimized simultaneously. Instead, we address implicit dependencies where resource consumption at a given workflow node may depend on the size of its input which in turn depend on the path followed by the workflow to reach this node. The multi-criteria optimization challenge is still there due to the dependencies. In [23, 24, 25] we developed multi-objective routing techniques for multimedia networks where node contributions to the path metric are *non-Markovian* (memory-ful) in nature. This is also the case in ARIA where depending on the properties of input objects an operator may take different actions.

1.2. Contributions of this Paper

In this paper, we introduce multi-sensor, single-actuator and multi-sensor, multi-actuator *workflow routing* for promoting creation and delivery of high precision, small delay, and small resource-usage media processing workflows. We term the adaptive flow management component *qStream*. The workflow in *qStream* is modelled as a flow graph, where nodes represent *scalable* sensors, filters, fusion operators, external sources, and actuators, whereas edges represent object flows. Service specifications include sensor-to-actuator delay, precision, resource, and frequency constraints. *It is essential that the workflow optimization executes in real-time. Therefore, due to real-time optimization requirements, we opt for heuristic solutions that provide very-close to optimal solutions* as demonstrated by the experiments in Section 6. In [1], we developed static optimization techniques for *chain* structured (single-sensor, single-actuator) media workflows and showed that even this relatively simple problem requires careful treatment due to the adaptive nature of the underlying middleware components. In this paper, we focus on the resource optimization of workflow plans for a larger class of workflow models, including tree-structured (multi-sensor, single-actuator) and directed acyclic (multi-sensor, multi-actuator) workflows. The algorithms presented in this paper address implicit dependencies where, for instance, resource consumption at a given node may depend on the size of its input which in turn may depend on the past transformations applied to the input objects in the workflow before reaching this node.

2. Background: ARIA Architecture

ARIA media processing and integration workflows describe how a specific media processing task can be performed by combining various sensors, media filter

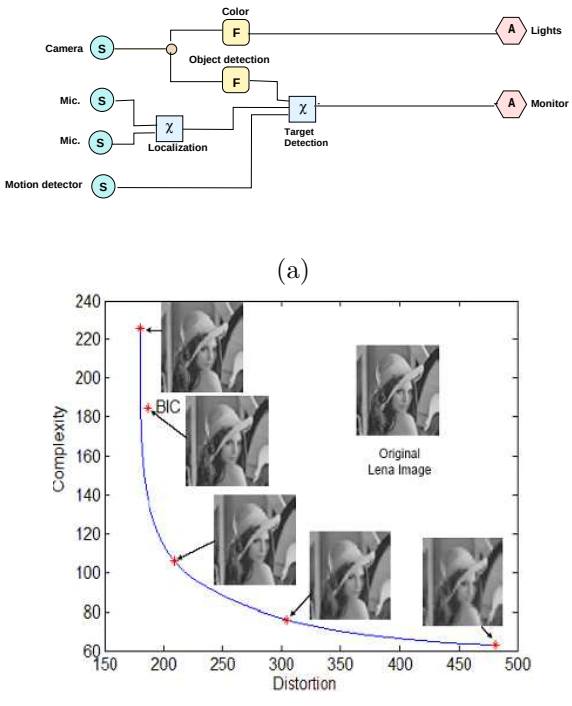


Figure 2. (a) An example *qStream* multimedia workflow: "S" denotes sensors, "F" denotes filters, "χ" denotes fusion operators, and "A" denotes actuators; (b) a scalable filter (the implementation of the filter and the example figure is provided by Prof. Hari Sundaram)

and information fusion operators, and actuators available [1, 2]. Unlike traditional workflows, a media service workflow captures (a) the continuous nature of the processing required at the nodes due to streaming nature of the sensory information, (b) inherent redundancy and imprecision in media, in terms of alternative ways of achieving a given goal, and (c) various quality of service requirements. In Figure 2(a), the vertices (or nodes) represent scalable sensors, filters, fusion operators and actuators, and edges represent connections that enable object flows between components.

ARIA Objects. The basic information unit is a data object. An object is produced by a sensor or an external data source. Depending on the task, an object can be as simple as a numeric value (such as the pressure values obtained from a surface sensor) or as complex as an image component segmented out from frames in a video sequence. $\mathcal{O}$ denotes the set of all object types.

ARIA Workflows. Each stream denotes a transmission channel of objects of the same type. For example, a sequence of 2-byte surface pressure values, measured

within 99.9% precision and generated every 10 milliseconds by a floor sensor, form an object stream.

Scalable ARIA Sensors. Sensors act as stream sources. For example, consider a motion sensor that can either provide high-precision motion information every 100 milliseconds or low-precision motion information every 10 milliseconds. This sensor is *scalable* and can make objects of type O available at different frequencies, sizes, and precisions.

Scalable ARIA Actuators. While sensors generate object streams, actuators consume object streams and map them to appropriate outputs. For example, consider a monitor that renders the 3D-mesh objects.

Scalable ARIA Filters. A filter takes an object stream as input, processes and transforms the object stream, and outputs a new stream of the transformed objects. For example, a module takes a stream of facial images as its input and returns a face signature vector as its output. Note that the precision of the result may depend on the number of consecutive faces considered or may depend on what type of a heuristic/neural-net is used. Consequently, filters (Figure 2(b)) are scalable and may provide multiple precisions, each with its own end-to-end delay and resource requirement.

Scalable ARIA Fusion Operators. A fusion operator is similar to a filter, except that it takes as its input multiple streams and returns as its output multiple streams. For example, consider a module which receives object-tracking information from multiple redundant sensors and outputs *fused* highly-precise object-tracking information. Fusion operators are also scalable; the total fusion delay, as well as other resources, would depend on the required precision.

2.1. *qStream*: ARIA Workflow Model with Alternate Behaviors

Within the ARIA middleware framework, a *qStream* workflow is an acyclic directed connected graph, $G(V, E, \beta)$:

- V is a set of nodes and E is a set of edges between the nodes on V.
- each node $v_i \in V$ has a set of **alternate** behaviors

$$\beta(v_i) = \{b_{1,i}, \ldots b_{j,i}\},$$

Intuitively, v_i has j levels of operation. ◇

Figure 3(a) provides an example. Each node in this example workflow is labelled with a number of alternative behaviors. Figure 3(b) shows a node with two behaviors and each behavior contains precision attributes.

Example 2.1 (Behavior of a motion filter) Consider a motion detector filter which receives a stream of images (a video stream) and identifies the motion direction of an object in this stream. Let the following describe the operation characteristics of this filter:

- input images are buffered in an input queue,
- w describes how many input images are to be used together to identify the motion direction,
- the filter operates in a pipelined manner; however it has a phase parameter, ϕ, which denotes the number of input images skipped between each computation,
- the frequency of the input images is f_{in},
- the size of each input image is s_{in},
- the queue drops one out of every λ inputs,

Based on these, we can see that

- since the queue drops one out of every λ input images, the effective input frequency of the filter is $f_{in} \times \frac{\lambda}{\lambda+1}$
- the operator needs enough input queue to hold w input images; therefore, the buffer resource requirement of the filter is $r = w \times s_{in}$,
- the filter computes one output for each ϕ input images, therefore the output frequency of the operator is $f = \frac{f_{in} \times \frac{\lambda}{\lambda+1}}{\phi}$
- the oldest input image used to generate an output has been in the queue until all the required $w - 1$ input images have been gathered in the queue; if we add the actual processing delay to this, we can compute the operation delay (the maximum delay observed by an input image before the corresponding output is generated) as $d = (w-1) \times \frac{1}{f_{in} \times \frac{\lambda}{\lambda+1}} + d_{processing}(s_{in})$

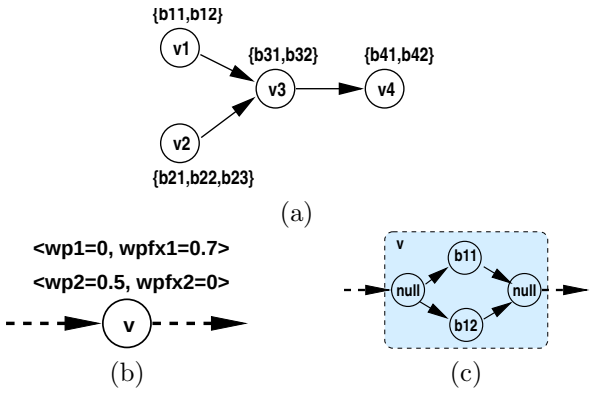


Figure 3. (a) An example workflow; (b) A node with two precision bevs; (c) the visual representation of the node

- the output precision of the operator is a function of the processing precision of the motion filter, the window size, the precision of the input objects, and the rate at with which inputs have been dropped from the queue before they are processed: $p = p_{in} \times p_{window}(w) \times p_{drop}(\lambda) \times p_{processing}$

- the size of the output objects, $sizeof(motion\_object)$, is fixed and independent of the input size. ◇

As illustrated in the above example, node behaviors can be described as a set of functions that operate on the characteristics (precision, size) of the objects in the workflow. Clearly, these functions depend on the implementation of the node as well as application semantics. In general, the node behavior functions for media operators include additive metrics (such as delay and jitter), multiplicative metrics (such as reliability and loss ratio), and concave/minmax metrics (such as bandwidth) [20]. Traditionally, within networking context, individual node/edge contributions are assumed to be independent of the path followed by the object before reaching the particular node. In various domains, however, node contributions to the overall path metric are memoryful in nature [23, 24, 25]. This is also the case in qStream; for instance, depending on the size of an in- coming object (which may depend on the precisions of the operators applied in earlier nodes) a node may require a different amount of time to operate.

Self-dependent Attributes: In Figure 3(b), the precision attribute of each behavior is self-dependent. The precision outcome of each behavior is represented with *relative* (for instance $w_{p,1}$ denotes that the output precision of a node is $w_{p,1}$ times the input precision) or *fixed* weights ($w_{p,fx,1}$ denotes that the output precision of the operator is fixed at $w_{p,fx,1}$, independent of the input). The first behavior in the example in Figure 3(b) will result in an output object with a fixed precision of 0.7, whereas the second behavior will result in an output with half of the precision of the input. If $w_p = 0$ this gives us a behavior with fixed precision outcome, whereas if $w_{p,fx} = 0$, this equation gives us a behavior with a relative precision. If a node has multiple input objects one alternative is to assume that the resulting precision is the *weighted average precision* of the inputs [26].

Dependent Attributes: Some attributes of behaviors depend on other attributes, complicating the overall planning and optimization process. For instance, the resource consumption of an ARIA node is a function of the input and output object sizes. The total amount of resources consumed at a given node, k, can be stated as a function of its input and output sizes.

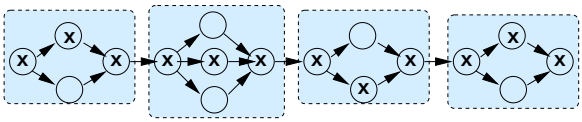


Figure 4. A chain of 4 nodes: the visual representation explicitly shows that each node has multiple behaviors.

3. Overview of Routing in Chain Workflows

In this section we provide an overview of chain workflows (with one sensor, one actuator, multiple adaptive filters, and no fusion operator) presented in [1]. In the following sections we will extend these results to workflows with fusion operators.

We denote chain workflows with $\mathcal{C}$. Figure 4 shows visual representation of a chain-like workflow (where a series of filters are applied on the sensor output). In this figure the behavior chosen for execution is marked with "X". These nodes constitute a path on the workflow chain. Given a path, $\rho = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n$, from the source, v_0, to the destination, v_n, we can calculate its various outcomes as follows [1]:

Precision outcome of a chain: If we denote the input precision to the path as p_{in}, we can summarize the precision outcome of the path, ρ, as

$$pre(\rho) = R_{p,\rho} \times p_{in} + F_{p,\rho},$$

where $R_{p,\rho}$ is the *combined* relative precision weight of the entire path ρ and $F_{p,\rho}$ is its fixed precision weight, i.e, precision transformations are self-dependent.

Size outcome of a chain: Size outcome of a chain depends on the size of the input object to the chain, as well as the size parameters of the nodes on the chain. Consequently, the size outcome of the path can be formulated similar to the precision outcome,

$$size(\rho) = R_{s,\rho} \times s_{in} + F_{s,\rho}.$$

Frequency outcome of a chain: Frequency outcome of a chain depends on the input frequency, as well as the frequency parameters of the nodes on the chain,

$$freq(\rho) = R_{f,\rho} \times f_{in} + F_{f,\rho}.$$

Resource consumption of a chain: Unlike the above self-dependant attributes, the resource consumption of a chain depends on the *object sizes* in addition to the resource consumption weights of the nodes on the chain, i.e. resource consumption is size-dependent,

$$res(\rho) = R_{r,\rho} \times \mathbf{s_{in}} + F_{r,\rho}.$$

$$
\begin{aligned}
res(\tau) = \; & res(c_0) + res(c_1) + \\
& res(c_2) + res(c_3) + \\
& res(c_4)
\end{aligned}
=
\begin{aligned}
& \left(\underbrace{(1.0)}_{\mathbf{s_{in,0}}} \times R_{r,\rho_0} + F_{r,\rho_0} \right) + \left(\underbrace{(1.0)}_{\mathbf{s_{in,1}}} \times R_{r,\rho_1} + F_{r,\rho_1} \right) + \\
& \left(\underbrace{(1.0)}_{\mathbf{s_{in,2}}} \times R_{r,\rho_2} + F_{r,\rho_2} \right) + \left(\underbrace{(R_{s,\rho_0} + F_{s,\rho_0} + R_{s,\rho_1} + F_{s,\rho_1})}_{\mathbf{s_{in,3}=s_{out,0}+s_{out,1}}} \times R_{r,\rho_3} + F_{r,\rho_3} \right) + \\
& \left(\underbrace{((R_{s,\rho_0} + F_{s,\rho_0} + R_{s,\rho_1} + F_{s,\rho_1}) \times R_{s,\rho_3} + F_{s,\rho_3} + R_{s,\rho_2} + F_{s,\rho_2})}_{\mathbf{s_{in,4}=s_{out,3}+s_{out,2}}} \times R_{r,\rho_4} + F_{r,\rho_4} \right)
\end{aligned}
$$

Figure 5. Resource usage of the example flow tree, τ, in Figure 6

Delay of a chain: Similar to resource consumption, the delay of a chain depends on the *object sizes*. Thus,

$$del(\rho) = R_{d,\rho} \times \mathbf{s_{in}} + F_{d,\rho}.$$

delay is size dependent.

For maximum-precision chain routing problem, the condition that enables Dijkstra-like optimizations hold and there exists an $O(E)$ greedy optimal algorithm. The maximum/minimum-frequency and maximum/minimum-size routing problems are similar and can be solved very efficiently.

Resource and delay requirements of chain workflows, on the other hand, depend on the sizes of the objects arriving at the nodes of the workflow. Objects may be filtered and fused, and object characteristics change depending on the workflow paths they follow. Among the variety of object characteristics we have to account for the size changes and their effects on resource consumption and delay. Optimal solutions to these single-sensor, single-actuator workflow problems require *expensive* non-linear mixed-integer programming (NLMIP). In [1] we have shown that carefully designed heuristic solutions can provide very-close to (within $1-2\%$ of) the optimal solutions in real-time.

4. Resource-adaptive Routing in Tree Workflows

In this section, we discuss optimization of tree-structured (multi-sensor, single-actuator) workflows. We use $\mathcal{T}$ to denote tree workflows. We represent a tree workflow, $\mathcal{T}$, in the form of a number of chains, $\mathcal{C}_i$ that are connected to each other in a hierarchical fashion (Figure 6). Consequently, a flow, τ, on a given tree workflow, $\mathcal{T}$, is a tree that is composed of paths (ρ_is) contributed by the individual chains ($\mathcal{C}_i$s). Most shortest path algorithms, like Dijkstra's shortest path algorithm, rely on the observation that "each subpath of a shortest path is a shortest subpath" and use this observation to eliminate non-promising paths. Unfortunately this condition does not always hold for the various behaviors of the *qStream* . For example, given the flow tree τ on the chain tree in Figure 6, we can compute its total resource usage $res(\tau)$ as shown in Figure 5. Here, each ρ_i corresponds to a path on the corresponding chain $\mathcal{C}_i$. The paths $\rho_0 \ldots, \rho_4$ form the tree τ. The figure shows that to choose a path, ρ_i, with the smallest resource consumption for a given chain, $\mathcal{C}_i$, one needs to know the sizes of the input objects, which could depend on the paths followed by the individual objects before reaching the chain $\mathcal{C}_i$. An optimal solution to this problem requires *expensive* NLMIP.

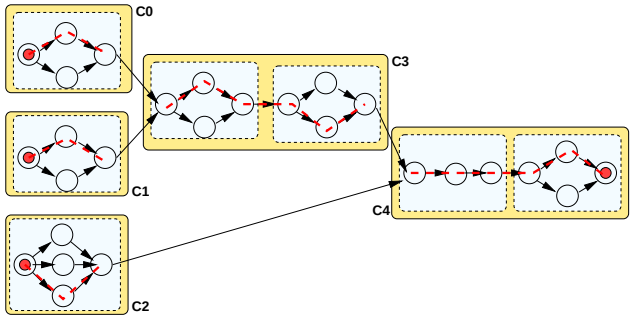
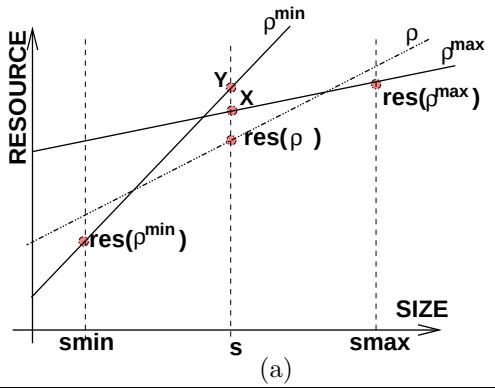


Figure 6. A tree workflow with multiple sensors, filter and fusion operators, and a single actuator

Let us reconsider the chain, $\mathcal{C}_4$, in the above example and assume that we have already calculated the maximum ($s_{in,4,max}$) and minimum ($s_{in,4,min}$) possible input object sizes to this chain. *qStream* would adapt the behavior of this chain to the size of the input. Let us denote the path parameters *qStream* would choose when provided with the largest objects, $R_{r,\rho_4^{max}}$ and $F_{r,\rho_4^{max}}$, for the worst-case scenario. Let us denote the parameters it would choose when provided with the smallest objects, $R_{r,\rho_4^{min}}$ and $F_{r,\rho_4^{min}}$, for the best-case scenario.



(a)

Low resource routing algorithm in tree workflows.

- **Input:** A directed acyclic graph $G(V, E, l)$ corresponding to a tree

1. $G^c(V^c, E^c) = chain\_tree(G)$
2. $G^{rc}(V^{rc}, E^{rc}) = reverse\_topologic\_sort(G^c)$
3. for all $(\mathcal{C}_i \in V^{rc})$ do
   (a) s_{min} is the minimum possible size of the input to chain $\mathcal{C}_i$
   (b) s_{max} is the maximum possible size of the input to chain $\mathcal{C}_i$
   (c) $\rho' = low\_resource\_path(\mathcal{C}_i, s_{max})$
   (d) $\rho'' = low\_resource\_path(\mathcal{C}_i, s_{min})$
   (e) calculate $exp(\Delta_{Y,X})$
   (f) if $exp(\Delta_{Y,X}) \geq 0$ then $path[i] = \rho'$
   (g) else $path[i] = \rho''$
4. Return the resulting paths for each chain in $G^c(V^c, E^c)$

(b)

Figure 7. (a) Best-/worst-case bounding of resource usage at a chain by the minimum and maximum input sizes and (b) Low resource routing

These worst- and best-case scenarios are depicted as two lines on the resource/size graph in Figure 7(a):

- *Worst-case resource consumption on $\mathcal{C}_4$:* Given these, we can compute the worst case resource consumption of the chain $\mathcal{C}_4$ as

$$res(\rho_4^{max}) = s_{in,4,max} \times R_{r,\rho_4^{max}} + F_{r,\rho_4^{max}},$$

where ρ_4^{max} is the path followed if the input size is equal to $s_{in,4,max}$. Then, for any feasible input size $s_{in,4}$, it follows that

$$res(\rho_4^{max}) \geq s_{in,4} \times R_{r,\rho_4^{max}} + F_{r,\rho_4^{max}}.$$

In Figure 7(a), the righthand side of this inequality is denoted as point X.

- *Best-case resource consumption on $\mathcal{C}_4$:* Similarly, if we consider the best case resource consumption of $\mathcal{C}_4$, we can see that

$$res(\rho_4^{min}) \leq s_{in,4} \times R_{r,\rho_4^{min}} + F_{r,\rho_4^{min}}$$

In Figure 7(a), the righthand side of this inequality is denoted as point Y.

Therefore, given an optimal resource usage plan for the entire tree, the resource usage $res(\rho_4)$ of chain $\mathcal{C}_4$ on this tree must satisfy the following condition:

$$res(\rho_4^{min}) \leq res(\rho_4) \leq Y, X \leq res(\rho_4^{max})$$

Using this inequality, we can define the amounts of error that would be caused if we used paths ρ_4^{max} or ρ_4^{min} instead of ρ_4 as

$$err_Y = Y - res(\rho_4) \quad \text{and} \quad err_X = X - res(\rho_4)$$

Obviously, without actually having computed the optimal tree yet and therefore not knowing the actual size of the input to the chain, we can not directly calculate err_Y or err_X. However, if we closely look at the error difference, $\Delta_{Y,X} = err_Y - err_X$, we can see that

$$\begin{aligned}
\Delta_{Y,X} &= err_Y - err_X \\
&= Y - res(\rho_4) - X + res(\rho_4) = Y - X \\
&= s_{in,4} \times R_{r,\rho_4^{min}} + F_{r,\rho_4^{min}} \\
&\quad - s_{in,4} \times R_{r,\rho_4^{max}} - F_{r,\rho_4^{max}} \\
&= s_{in,4} \times (R_{r,\rho_4^{min}} - R_{r,\rho_4^{max}}) + \\
&\quad (F_{r,\rho_4^{min}} - F_{r,\rho_4^{max}})
\end{aligned}$$

In the above, $s_{in,4}$ denotes the size of the input object to the chain $\mathcal{C}_4$ on the resource optimal tree. Since we already know that any feasible input size must be between $s_{in,4,min}$ and $s_{in,4,max}$, the expected difference in error, $exp(\Delta_{Y,X})$, can be calculated as

$$\begin{aligned}
exp(\Delta_{Y,X}) &= \frac{1}{(s_{in,4,max} - s_{in,4,min})} \int_{s_{in,4,min}}^{s_{in,4,max}} (s \times \\
&\quad (R_{r,\rho_4^{min}} - R_{r,\rho_4^{min}}) + (F_{r,\rho_4^{min}} - F_{r,\rho_4^{max}})) \, \delta s \\
&= \frac{1}{2} \cdot (R_{r,\rho_4^{min}} - R_{r,\rho_4^{max}}) \times \\
&\quad (s_{in,4,max} - s_{in,4,min}) + (F_{r,\rho_4^{min}} - F_{r,\rho_4^{max}})
\end{aligned}$$

If $exp(\Delta_{Y,X}) \geq 0$, the expected error due to the use of ρ_4^{min} (best-case) is greater than the expected error in using ρ_4^{max} (worst-case); therefore, substituting ρ_4 with ρ_4^{max} is more desirable. Otherwise, substituting ρ_4 with ρ_4^{min} leads to a smaller expected error.

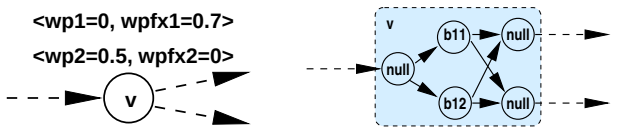
In Figure 7(b), we use this observation to develop an $O(V^2E)$ *heuristic* algorithm for low resource routing in tree workflows. Low resource routing algorithm firstly computes the minimum and maximum input sizes to all chains. Then, for each chain, based on the corresponding value of the expected error, $exp(\Delta_{Y,X})$, it chooses the appropriate execution plan.

5. Resource-adaptive Routing in Directed Acyclic Workflows

In [24], we developed a resource-adaptive routing algorithm for active networks. The algorithm in [24] considers the largest (and smallest) cost change that a unit increase in the size of an object being delivered through a given network node can cause and applies this in the edge relaxation policy to decide the direction of routing. The strategy is analogous to the steepest decent search method in a continuous solution space. In this section, we built on this obervation to develop a resource-adaptive routing scheme for directed acyclic (DA) workflows and extended the stategy by computing an upper bound and a lower bound.

First, given a unit size of object generated at each sensor, the largest and smallest input object sizes that an intermediate node can expect are calculated respectively at each node in the workflow. We label the largest and smallest input object sizes of node i as s_i^{max} and s_i^{min}. We then compute the resources consumed at each intermediate node in the workflow for both, s_i^{max} and s_i^{min}, respectively. The smallest and largest object size estimate break the dependency on the traversed path for the calculation of the resource consumption at each node in the workflow. Thus, we can compute the lower- and upper-bounds on the total resource consumption of the workflow using these estimates.

Unlike the tree-networks, in general directed acyclic workflows, nodes can multiplex the output objects to several receiving nodes. Furthermore, in ARIA, their destination nodes can have different levels of Quality of Service requirements on their outputs (Figure 8). To deal with the resulting complexity, we decompose the routing problem into a set of sub-workflows where only one actuator is present in the sub-workflows; each sub-workflows is a multi-sensor, single-actuator tree as discussed in the previous section. Thus, for each sub-workflow, the upper bound and lower bound of resource consumption of a sub-workflow can be calculated as discussed earlier and the corresponding behavior is marked. Next, for each sub-workflow, nodes of a sub-workflow are traversed in a topological order and the differences between the resource consumption using the actual input object size and the pre-computed bounds are calculated. Similar to the tree optimization algorithm in Section 4, the behavior which corresponds to the bound with the smaller difference is selected. At the end of the step, for each node, we sum up its resource consumption. Finally, we identify the most costly actuator, fix paths for that actuator and visit all actuators in the decreasing order of expected cost until the paths to all actuators are taken.



(a) (b)

Figure 8. (a) a node with two behaviors and two outputs; (b) the visual representation of the node

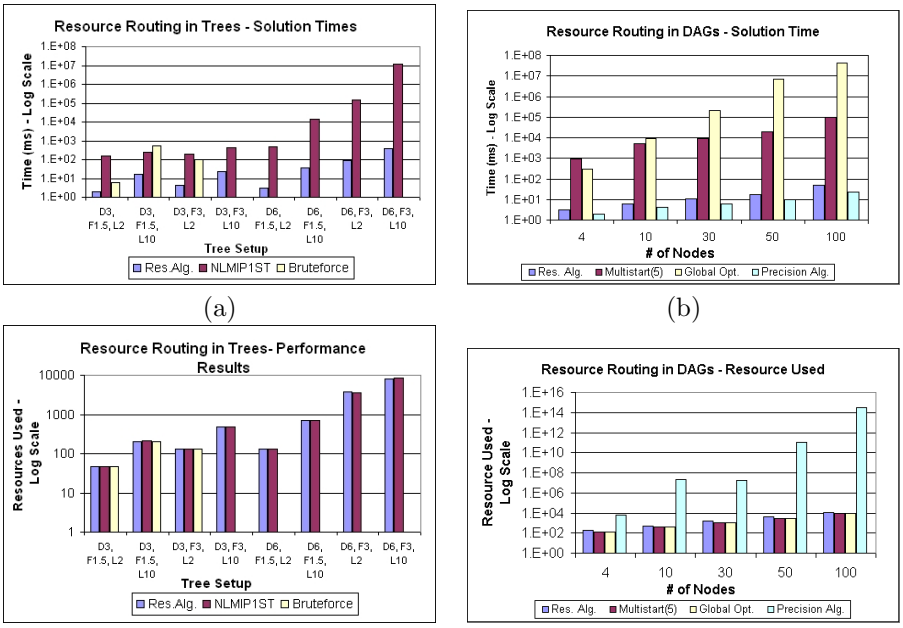
6. Experimental Evaluation

For our evaluating the efficiency and effectiveness of these algorithms, we created data sets corresponding to various *qStream* tree workflows:

Experimental Workflow Properties	
Chain length	2. . . 50 nodes
Tree depth	2. . . 6
Avg. Tree fanout	1.5. . . 3
Number of behaviors per node	2. . . 4
Precision weight	0.5. . . 1.0
Size weight	0.2. . . 2.0
Resource weight	1.0. . . 10.0
Delay weight	1.0. . . 10.0

In order to observe and interpret the performance of the proposed optimization mechanisms, in addition to our heuristics, we used two alternative optimization approaches for comparisons: NLMIP using LINGO optimization software and *bruteforce* enumeration of all alternatives. The bruteforce enumeration enabled us to obtain not only the best solution, but also the worst and average solutions, allowing us to observe the relative performance of our techniques while NLMIP enabled the comparison for larger workflows.

To analyze the performance of the resource-adaptive routing algorithm in DA workflows, we modified the random network generator in [27]. The modified network generator starts with a chain of n nodes, randomly adds and removes arcs provided the graph is still acyclic and connected. For the purpose of the performance comparison, we varied the number of nodes, n, in the staring chain to create various *qStream* DA workflows. We formulated Non-linear Integer Programming Models in LINGO to compare the relative performance of our algorithms with the results that Multistart Solver and Global Solver of LINGO generated. Multistart(5) restarts the LINGO built-in heuristic algorithm 5 times from different initial points and takes the best result as the optimization solution. LINGO's Global solver takes use of range bounding and range reduction techniques within a branch-and-bound framework to find global solutions. The algorithms were implemented in Java and ran on Redhat 7.2 Linux workstations, with 1 GB RAM, 1.8 GHz processor. LINGO with Nonlinear Global Solver was installed on a Windows PC, with 256 MB RAM, 2.0 GHz processor.



- **NLMIP1ST stands for the 1st local optima for the NLMIP; i.e. the graphs in this figure reports the 1^{st} local optima for the NLMIP.**
- **D stands for the tree depth.**
- **F stands for the average fanout.**
- **L stands for the chain length.**
- **Bruteforce results are omitted where executions took more than a day.**

Figure 9. (a,b) resource routing algs on tree and directed acyclic workflow multiple orders faster than NLMIP and (c,d) provide near optimal (within 1-2%) results.

6.1. Efficiency Analysis

Resource Routing in Tree Workflows: Obviously the bruteforce approach did not scale beyond exceedingly small workflows. For larger workflows, we solved the non-linear mixed integer formulation of the problem and thus were able to make comparisons against the optimal solutions. However, NLMIP did not scale well either. For instance, for a tree of depth 4, average fanout 2, and chain length 10, the global resource optimum was achieved in 33 seconds and the global delay optimum was achieved in 23 seconds using LINGO. For larger workflows, global optimum took anywhere between tens of minutes (45 mins 3 seconds for a tree with 5 depth, 2 average fanout, and chain length 10) to many hours, in contrast to less than a second required by our heuristic. *Therefore, while generating the NLMIP solutions for comparison, in tree experiments, we settled for the first solution (most of the time a local optimum) found by LINGO.* This took anywhere from 30 minutes to several hours to execute, still unusable in real settings, but significantly faster than bruteforce optimization as well as finding the global optimal of NLMIP problems. As Figure 9(a) shows, the *qStream* algorithms are multiple orders faster than the bruteforce and NLMIP.

Resource Routing in DA Workflows: In this experiment we scaled the size of the experimental workflows. From Figure 9(b), we see that the resource-adaptive routing algorithm presented in this paper executes within 100 ms for different sizes of workflows we experimented. In contrast, the multistart and global solver operated consistently three to four orders slower. Figure 9(b) also include the optimization times required for a self-dependent attribute (precision). As this figure shows, the proposed algorithm for resource optimization operates on the same order as the time required for optimizing this self-dependent attribute.

6.2. Effectiveness Analysis

Resource Routing in Tree Workflows: Our resource routing algorithm in tree workflows is sub-optimal. However, the results returned by our algorithm were always within 2.0% of the optimal as shown in Figure 9(c) and most of the time better than the first local optima returned by LINGO. Whereas our algorithm ran for less than 30ms for a workflow of depth 4, avg fanout 2, and chain length 10, the bruteforce algorithm ran for more than 2hrs for the same configuration, whereas NLMIP required 33 seconds. In general, our algorithm returns near-optimal results and is multiple orders faster than both alternatives (Figure 9(a)).
Resource Routing in DA Workflows: Figure 9(d) shows the resource costs of various algorithms. Solutions returned by the propose algorithm is within 1-2% to the optimal solutions that Global Solver generates. In contrast, a resource-agnostic optimizer could

cost a significantly higher resources as depicted by the resource behavior of the precision optimizer.

Summary: The experiments, as depicted in Figures 9(a) through (d), showed that the proposed heuristic algorithms work efficiently (under 100ms) and effectively (within 1-2% to the optimal). They scale better than other formulations of the same problem, yet, return almost optimal results within a time-frame suitable for real-time system.

7. Conclusions and Future Work

ARIA is a media workflow architecture, designed primarily for real-time interactive art applications. ARIA operators filter various features from streamed data, fuse, and map media streams onto output devices. The nodes in the workflow correspond to adaptive computing elements. We focused on this adaptive nature of the workflows and presented algorithms that efficiently identify small-resource tree (multi-sensor, single-actuator) and direct acyclic (multi-sensor, multi-actuator) workflows. For future work we will investigate the resource-adaptive routing with explicit quality constraints. To handle the quality constraints, we opt for defining a penalty function to combine (1) minimization of the resource consumption and (2) satisfaction of the quality constraints. We will also develop quality-adaptive runtime correction techniques which rely on the presented heuristics. These complement the optimizer by modifying plans at runtime.

Acknowledgements

The authors would like to thank Prof. Maria Luisa Sapino for her feedback on the ARIA media workflow model and Prof. Hari. Sundaram for his input on the scalable media processing operators.

References

[1] K.S. Candan, L. Peng, K.D. Ryu, K.S. Chatha and C. Mayer. *Efficient Stream Routing in Quality- and Resource-Adaptive Flow Architectures,* Intl. Workshop on Multimedia Information Systems, 2004.

[2] L. Peng, K.S. Candan, K.D. Ryu, K.S. Chatha and H. Sundaram. *ARIA: An Adaptive and Programmable Media-flow Architecture for Interactive Arts,* ACM MM Interactive Arts Program, 2004.

[3] http://www.cycling74.com/index.html

[4] http://www-crca.ucsd.edu/∼msp/Pd_documentation/

[5] X. Gu and K. Nahrstedt. *Distributed Multimedia Service Composition with Statistical QoS Assurances,* IEEE Transactions on Multimedia, 2005.

[6] B. Raman and R. H. Katz. *An architecture for highly available wide-area service composition,* Computer Communication, 26(15):1727–1740, September 2003.

[7] B. Raman and R. H. Katz. *Load Balancing and Stability Issues in Algorithms for Service Composition,* IEEE INFOCOM, 2003.

[8] D. Xu and K. Nahrstedt. *Finding Services Paths in a Media Service Proxy Network,* MMCN, 2002.

[9] X. Fu, W. Shi, A. Akkerman and V. Karamcheti. *CANS: Composable, Adaptive Network Services Infrastructure,* Proc. of USITS, March, 2001.

[10] A. Black, J. Huang, R. Koster, J. Walpole and C. Pu. *Infopipes: An abstraction for multimedia streaming,* Multimedia Systems 8: 406-419, 2002.

[11] C. Lee, J. Lehoczky, R. Rajkumar and D. Siewiorek. *On Quality of Service Optimization with Discrete QoS Options,* IEEE RTSS, 1999.

[12] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar and J. Hansen. *A Scalable Solution to the Multi-Resource QoS Problem,* IEEE RTSS, 1999.

[13] F. Chang and V. Karamcheti. *A Framework for Automatic Adaptation of Tunable Distributed Applications,* Cluster Computing, Vol. 4, Issue 1, May 2001.

[14] D. Xu, K. Nahrstedt and D. Wichadakul. *QoS and Contention-Aware Multi-Resource Reservation,* Cluster Computing, 4: 95-107, 2001.

[15] K. Nahrstedt and J. M. Smith. *The QOS Broker,* IEEE MultiMedia, Vol. 2, Issue 1: 53 - 67, 1995.

[16] N. Wang and G. Pavlou. *DQM : An Overlay Scheme for Quality of Service Differentiation in Source Specific Multicast,* workshop on QoS-IP, 2003.

[17] M.A.Shah and S.Chandrasekaran. *Fault-Tolerant, Load-Balancing Queries in Telegraph,* SIGMOD Record, v.30 n.2, p.611, June 2001.

[18] D.Carney*et.al. Monitoring Streams-A New Class of Data Managment Applications.* VLDB02.

[19] D.H.Lorenz and A.Orda. *QoS Routing in Networks with Uncertain Parameters*,INFOCOM98.

[20] Z.Wang and J.Crowcroft. *Quality of Service Routing for Supporting Multimedia Applications*, IEEE JSAC vol.14,no.7,Sep.96.

[21] S.Siachalou and L.Georgiadis. *Efficient QoS Routing*, INFOCOM 2003.

[22] T.Korkmaz and M.Krunz. *Multi-Constrained Optimal Path Selection.* INFOCOM 2001.

[23] A. Sen, K. S. Candan *et.al. On Shortest Path Problems with "non-Markovian" Link Contribution to Path Lengths.* NETWORKING 2000: pp. 859-870.

[24] K. S. Candan and Y. Yang. *Least-Cost High-Quality Object Retrieval for Distributed Multimedia Collaborations.* ICMCS99.

[25] K.S. Candan *et al. Collaborative Multimedia Systems:Synthesis of Media Objects.* IEEE TKDE, Vol.10,Nr.3,pp.433-457, May-Jun98.

[26] K.S. Candan and W.-S. Li. *On Similarity Measures for Multimedia Database Applications.* Knowledge and Info. Sys. 3(1): 30-51,2001.

[27] J.S. Ide and F.G. Cozman. *Random Generation of Bayesian Networks.* Brazilian Symp. on AI, 2002.