Efficient Stream Routing in Quality- and Resource-Adaptive Flow Architectures

K. Selçuk Candan Lina Peng Kyung D. Ryu Karamvir S. Chatha Christopher Mayer
Computer Science and Engineering Dept.,
Arizona State University, Tempe, AZ, 85287, USA.
{candan,lina.peng,kdryu, kchatha,chris.mayer}@asu.edu

Abstract

In this paper, we address challenges faced when optimizing data and object streams in the presence of quality and resource *adaptable operators on the flow network. We present the qStream* [1] *flow architecture that processes, filters, and fuses sensory inputs and actuates responses in real-time while providing various Quality of Service (QoS) guarantees. The components of the architecture are programmable and adaptable; i.e, the delay, size, frequency, and quality/precision characteristics of individual operators can be controlled via a number of parameter values. Therefore, each data object streamed between components is subject to transformations and quality changes based on these parameters. The quality of an output data object and the corresponding delay and resource usage depend on the values assigned to parameters of the operators in the object flow path. In this paper, we develop performance models for the quality-adaptive stream optimization. We introduce a class of* route optimization problems *that promote creation and delivery of high precision, small delay, and small resource-usage objects to the actuators. We develop QoS, resource, and deadline aware flow optimization techniques. We experimentally show the effectiveness of the algorithms.*

1 Introduction

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, determining their relative spatial positions, and detecting certain simple events. Interactive performances that use this stage requires an information architecture that processes, filters, and fuses sensory inputs and actuates audio-visual responses in real-time, while providing appropriate QoS guarantees.

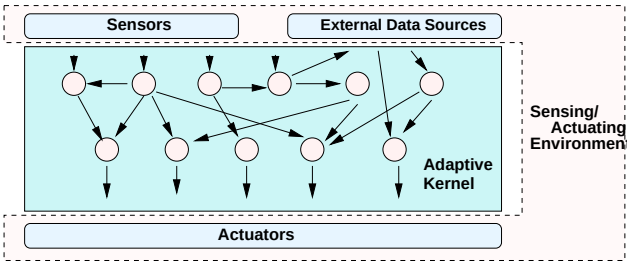


Figure 1. *qStream* flow framework

In this paper, we present the quality- and resource-adaptive *qStream* flow architecture that processes, filters, and fuses sensory inputs and actuates responses in real-time while providing certain Quality of Service (QoS) guarantees. As shown, in Figure 1, *qStream* is deployed in an environment equipped with sensors (such as video cameras, motion detectors and trackers, microphone arrays, and pressure sensors) and actuators that operate on various types of objects. Consequently, unlike most existing work in stream processing which assume relatively small and uniform tuples in the data streams, our focus is on objects (such as sensed media objects) that come inherently in varying sizes, qualities, and resource requirements. The *qStream* flow architecture provides a *quality-adaptive* flow network, consisting of various adaptive filter and fusion operators. The quality-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

[1]This work is part of the Architecture for Interactive Arts (ARIA) project supported by NSF Grant # 0308268

data object streamed between components are, therefore, subject to changes based on these parameters. The quality of an output data object and the corresponding delay and resource usage depend on parameters assigned to the operators in the object's filter and fusion path.

Recently there has been a flurry of activities in the area of data stream management. Due to the continuous nature of data streams, adaptation has been a crucial aspect of these systems. Telegraph [1] provides an adaptive dataflow engine that moves data through a number of database operators (such as selection and join). Aurora [3], focuses on QoS- and memory-aware operator scheduling and semantic load shedding for coping with transient spikes in a data stream network. *qStream* routing takes place in two phases. In the first phase a static *qStream* optimizer generates a set of query plan alternatives and registers them with the system. A runtime correction program in the second phase dynamically selects appropriate flow plan alternative or reoptimizes if needed. *Therefore, it is essential that the optimization executes in real-time.*

1.1 Contributions of this Paper

There are often tradeoffs between various desired characteristics, such as least delay and high quality/precision. Optimization problems with trade-offs between objectives are generally intractable. In this paper, we introduce *flow routing problem*s that promote creation and delivery of high precision, small delay, and small resource-usage objects to the actuators. As in many data flow architectures, such as Aurora [3], the *qStream* flow architecture is modelled as a flow network, where vertices (or nodes) represent sensors, filters, fusion operators, external sources, and actuators, whereas edges represent object streams between components (Figure 2(a)). The operator specifications used by the *qStream* scheduler and optimizer include (1) alternative characteristics of the sensors (Figure 2(b)); (2) precisions and computational overheads of the filters (such as object feature extractors); (3) precisions and computational overheads of the fusion operators; and (4) characteristics of the actuators. QoS specifications include sensor-to-actuator delay, precision, resource, and frequency constraints. In most cases, optimal solutions to these problems require *expensive* nonlinear mixed-integer optimizations. *Instead, 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 4.

2 *qStream* Flow Architecture

qStream is modelled as a network where the set of vertices (or nodes) represents the set of sensors, filters, fusion operators, and actuators; the set of edges represents connections that stream objects between components (Figure 2).

2.1 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 an integer denoting the pressure applied on a surface sensor) or as complex as an image component segmented out from frames in a video sequence. Each object streamed through *qStream* contains *an object payload*, such as a string, a numeric value, or an image region, and *a meta-data header*, describing the object properties:

- *size* of the object data that defines the memory or buffer requirement for the object,
- *precision* of the object data[2],
- a *set of resource usage stamps* and a *set of timestamps* acquired by the object as it goes through various operators.

We denote the set of all object types using $\mathcal{O}$.

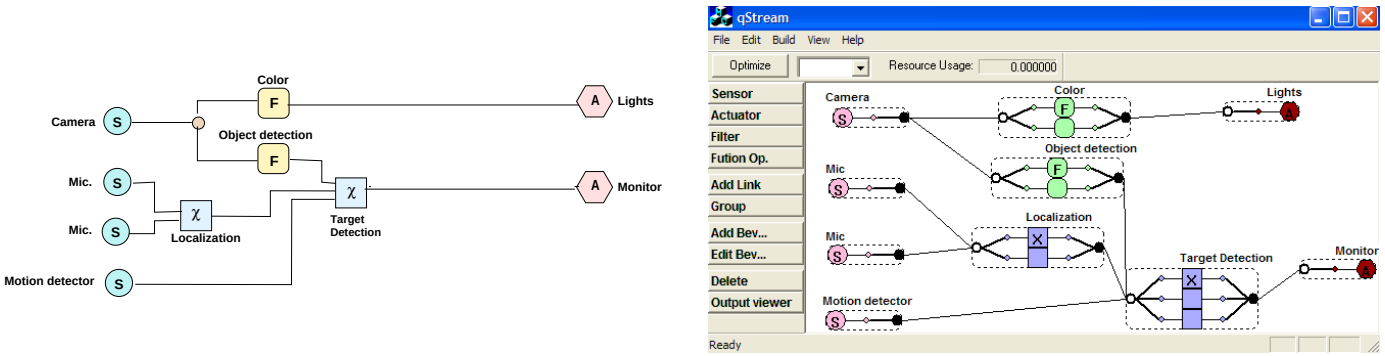
2.2 Streams

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, forms an object stream. Consequently, given an object type $O \in \mathcal{O}$, we can represent a *stream type* as a quadruple $\langle O, f, b, p \rangle$, where f is the frequency with which data is available in the stream, b is the resource requirement of each object, and $p \in [0, 1]$ is the precision of the data in the stream.

2.3 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*: a scalable sensor can make objects of type O available at different frequencies, sizes, and precisions.

[2]For different object types, precision may mean different things; for an image, its precision may mean its resolution, whereas for a coordinate value, it may mean the level of confidence provided by the object tracking sensors.



(a) (b)

Figure 2. (a) An example flow network: "S" denotes sensors, "F" denotes filters, "χ" denotes fusion operators, and "A" denotes actuators; (b) the corresponding flow network created in *qStream* GUI (note that GUI shows that some of the nodes have multiple/alternative implementations)

2.4 Actuators

While sensors generate object streams, actuators consume object streams and map them to appropriate outputs. For example, consider a monitor that can render the 3D-mesh objects delivered to it.

2.5 Filters

A filter takes an object stream as input, processes and transforms the objects in the stream, and outputs a new stream consisting of the transformed objects. For example, consider a module that takes a stream of facial images as its input and returns a face signature vector as its output. This module is a transforming filter. 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, the filter may provide multiple precisions, each with its own end-to-end delay and resource requirement and delay.

2.6 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. The total fusion delay, as well as other resources, would depend on the degree of the required precision.

2.7 *qStream* Network

We model the *qStream* flow architecture in the form of a directed connected graph.

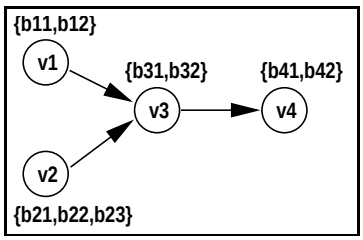


Figure 3. An example *qStream* flow network

Definition 2.1 (*qStream* **network**) *A qStream flow network is an acyclic directed connected graph,* $G(V, E, \beta)$*, where*

- V *is a set of nodes and* E *is a set of edges between the nodes on* V*.* $S \subset V$ *is the set of* sensor *nodes. The nodes in* S *do not have incoming edges; i.e., they are sources.* $A \subset V$ *is the set of* actuator *nodes. The nodes in* A *do not have outgoing edges; i.e., they are sinks.*
- *an object state is a 5-tuple,* $\langle s, r, f, p, d \rangle$*, where* s *denotes object size,* r *denotes the total resource consumption since the beginning of the corresponding flow,* f *denotes the frequency,* p *denotes the precision, and* d *denotes the total delay observed since the beginning of the corresponding flow. The set of all object states is* $\mathcal{OS}$*.*
- *a behavior of a node with* l *input edges is a function,* $o : \mathcal{OS}^l \rightarrow \mathcal{OS}$*, which takes* l *input object states and returns one output object state. Intuitively, the output of a node depends on the states of the incoming objects as well as the configuration of the node itself.*
- *each node* $v_i \in V$ *has a set of* **alternate** *behaviors*

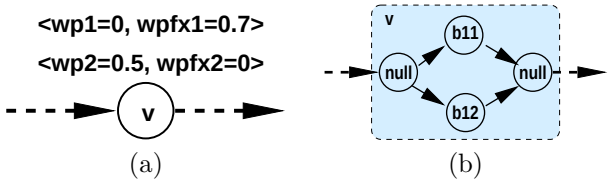


Figure 4. (a) A node with two precision behaviors; (b) the ABNs of the node

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

where each behavior is described in terms of the output behavior of the node. Intuitively, v_i has j different levels of operation. ◇

Figure 3 provides an example. Each node in the network is labelled with a number of alternative behaviors.

2.8 Node Behavior

The node behavior is a collection of size, resource, frequency, precision, and delay behaviors that collectively describe the operation characteristics of a given *qStream* node (Figure 2(b)). The node behavior 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) [12]. In tuple routing, on the other hand, tuple sizes are linearly modified by selection, projection, and join operators; however, a percentage of tuples may be dropped if necessary [4]. If these tuples are fuzzy, then *arithmetic average* (as well as *min* and *product*) can be used as a merge function to define fuzzy relational operators [18]. Traditionally, 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 *memory-ful* in nature [15, 16, 17]. This is also the case in *qStream* ; for instance, depending on the size of an incoming 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.

Precision behavior: The output precision of a node (e.g., a filter) may either be independent of the properties of the input objects or be a function of their precisions. Figure 4(a) shows a node with two behaviors. In this example, each behavior consists of *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 4(a) will result in an object with a fixed precision of 0.7, whereas the second behavior will result in an output object with half of the precision of the input object.

For only representational convenience, *without any loss of generality*, we combine the fixed and relative components of the precision behavior using a linear combination of the fixed and relative precision weights, such that if $w_p = 0$ this gives us the fixed behavior, whereas if $w_{p,fx} = 0$, this equation gives a relative behavior:

$$p_{out} = w_p \times \left(\sum_{o_i \in inputobjects} p_{in,i} \right) + w_{p,fx},$$

Here, $p_{in,i}$ is the precision of the i^{th} input object. Note that, if a node has multiple input objects, this model assumes that the resulting precision is a function of the *average precision* of the inputs [18].

Size behavior: Similarly to the precision behavior, we model the size behavior of a node as

$$s_{out} = w_s \times \left(\sum_{o_i \in inputobjects} s_{in,i} \right) + w_{s,fx},$$

where $s_{in,i}$ is the size of the i^{th} input object.

Frequency behavior: Similarly to the precision and size behaviors, the frequency behavior of each node is a function of its input frequencies:

$$f_{out} = w_f \times \left(\sum_{o_i \in inputobjects} f_{in,i} \right) + w_{f,fx},$$

where $f_{in,i}$ is the frequency of the i^{th} input object.

Resource behavior: The resource consumption of a node is a function of the input and output object sizes. The total amount of resources consumed at a given node can be stated as

$$\underbrace{w_{r,in} \times \left(\sum_{o_i \in inputobjects} s_{in,i} \right) + w_{r,out} \times s_{out} + w_{r,fx}}_{\text{resources consumed at this node}}$$

The total (end-to-end) resources consumed for the creation of an object can be calculated by summing up the resources used by the operator which output the object and the resources consumed earlier to compute the inputs to the operator. Thus, r_{out}, which denotes the total amount of resources consumed in the flow network for the output object of a given node can be stated as

$$r_{out} = resources\ at\ this\ node + \underbrace{\sum_{o_i \in inputobjects} r_{in,i}}_{\text{resources consumed earlier}}$$

where $r_{in,i}$ is the resources consumed by the i^{th} input.

Delay behavior: The delay behavior of each node is a function of its input sizes and output size:

$$\underbrace{\left(\sum_{o_i \in inputobjects} w_{d,in} \times s_{in,i}\right) + w_{d,out} \times s_{out} + w_{d,fx}}_{\text{delay due to this node}}$$

Delay is additive in nature. The overall (end-to-end) delay observed by an object can be calculated by summing up the delay introduced by the operator, which output the object, and the *maximum delay* observed by the input objects to the operator. Thus, d_{out}, which denotes the time consumed in the flow network for the output object of a given node can be stated as

$$d_{out} = delay\ at\ this\ node + \underbrace{max\{d_{in,i} | o_i \in inputobjects\}}_{\text{delay due to earlier nodes}},$$

where $d_{in,i}$ is the delay observed by the i^{th} input.

2.9 Nodes with Alternative Behaviors

In *qStream* each node can have alternative behaviors (Definition 2.1, Figure 4 (a)). Each *qStream* flow node can also be represented as a set of vertices and edges, such that each vertex has one and only one behavior. In the graph transformed as such, each vertex is called an *alternative behavior node* (ABN) as it represents a single alternative behavior of the corresponding operator. Figure 4 (b) shows the two ABNs that correspond to the single flow node shown in Figure 4 (a). Note that in addition to the two ABNs, the figure also contains two *null* ABN vertices whose job is simply to connect other behavior vertices. The *null* ABN has the following behavior:

- $\langle w_p = 1, w_{p,fx} = 0\rangle$, $\langle w_s = 1, w_{s,fx} = 0\rangle$, $\langle w_f = 1, w_{f,fx} = 0\rangle$, $\langle w_{r,in} = 0, w_{r,out} = 0, w_{r,fx} = 0\rangle$, $\langle w_{d,in} = 0, w_{d,out} = 0, w_{d,fx} = 0\rangle$

3 Optimization of Flow Routes

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. This enables these algorithms to identify shortest paths very efficiently. Unfortunately this condition does not always hold for the various behaviors of the *qStream* as explained in Section 3.1.

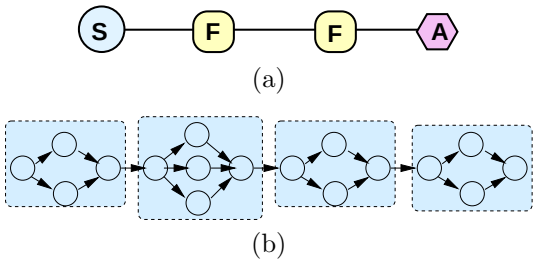


Figure 5. (a) An *qStream* chain and (b) its ABNs

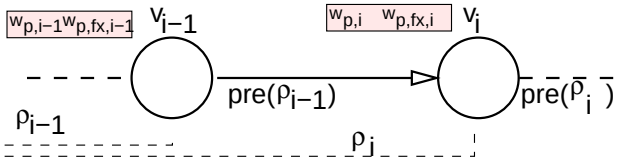


Figure 6. Precision outcome depends on the precision of the input and the node's precision weights

We discuss optimization of increasingly complex flow graphs. We do optimizations on the equivalent ABN graph instead of the original *qStream* network. We start with the chain networks (with one sensor, one actuator, multiple adaptive filters, and no fusion operator) and we extend the results to tree networks with fusion operators.

3.1 Routing in Adaptive Chain Networks

Figures 5(a) and (b) show a chain-like network where a series of filters are applied on the sensor output and its ABNs. We denote chain networks $\mathcal{C}$.

3.1.1 Path Outcomes on Chain Networks

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, on a given chain $\mathcal{C}$, we can calculate its various outcomes as follows:
Precision outcome: Let $\rho_{0:i}$ (or simply ρ_i) denote the subpath from source, v_0 to node v_i. The precision outcome of node v_i depends on the precision of its input object as well as this node's precision weights:

$$pre(\rho_i) = pre(\rho_{i-1}) * w_{p,i} + w_{p,fx,i},$$

Consequently, if we denote the input precision to the path as p_{in}, we can summarize the precision outcome of the path, $\rho(= \rho_{0:n})$, as

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

where $R_{p,\rho}$ is the combined relative precision weight of the path ρ and $F_{p,\rho}$ is its fixed precision weight.
Size outcome: Size outcome of a node depends on the size of the input object, as well as the node's size

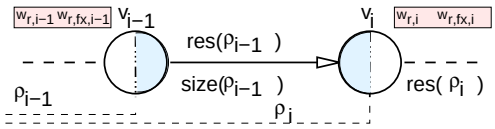


Figure 7. An object consumes resources both at the source and at the destination

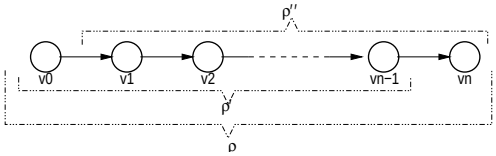


Figure 8. Path, ρ, and its prefix, ρ', and tail, ρ''

weights. 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: Frequency outcome of a node depends on the input frequency, as well as the node's frequency weights:

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

Resource consumption: Unlike the precision outcome of a node, which depends only on the input precision and precision weights of the node, the resource consumption of a node depends on the *input/output object sizes* in addition to the node's resource consumption weights (Figure 7):

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

Delay: Since there are no fusion operators on a chain network, each node has at most one input; hence, there is no need to consider the *maximum* delay of the inputs. Thus,

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

3.1.2 Precision, Frequency, and Size Routing in Chain Networks

For these cases it is easy to see that the condition that enables Dijkstra-like optimizations hold and, relying on this observation, we can develop an optimal algorithm that finds the highest precision path on a given a chain. In fact, we can develop an $O(E)$ greedy optimal algorithm that leverages the special structure of the chain graph. Note also that the maximum-frequency, minimum-frequency, maximum-size, and minimum-size routing problems are similar to the high precision routing; hence they can be solved very efficiently.

$$\underbrace{\sum_{k=1}^{n}\left((w_{r,out,k-1}+w_{r,in,k})\times\prod_{i=0}^{k-1}w_{s,i}\right)}_{A_\rho}+$$
$$\underbrace{\sum_{k=1}^{n}\left((w_{r,out,k-1}+w_{r,in,k})\times\sum_{i=0}^{k-1}\left(w_{s,fx,i}\times\prod_{j=i+1}^{k-1}w_{s,j}\right)\right)+\sum_{k=0}^{n}w_{r,fx,k}}_{B_\rho}$$

(a) Resource consumption of a path

$$\underbrace{(w_{r,out,0}+w_{r,in,1})\times(w_{s,0}+w_{s,fx,0})+w_{r,fx,0}}_{\alpha_{0:1}}+$$
$$\underbrace{(w_{s,0}+w_{s,fx,0})}_{\sigma_{0:0}}\times\underbrace{\sum_{k=2}^{n}\left((w_{r,out,k-1}+w_{r,in,k})\times\prod_{i=1}^{k-1}w_{s,i}\right)}_{A_{\rho''}}+$$
$$\underbrace{\sum_{k=2}^{n}\left((w_{r,out,k-1}+w_{r,in,k})\times\left(\sum_{i=1}^{k-1}w_{s,fx,i}\times\prod_{j=i+1}^{k-1}w_{s,j}\right)\right)+\sum_{k=1}^{n}w_{r,fx,k}}_{B_{\rho''}}$$

(b) Res. cons. of the same path in terms of the cons. of its tail

Figure 9. Recursive subpath property (between a path and its tail) used for resource optimization

3.1.3 Resource and Delay Routing in Chain Networks

Resource and delay requirements depend on the sizes of the objects arriving at the nodes of the flow network. Since objects may be modified by the operators within the flow network, we have to account for the size changes and their effects on resource consumption and delay. Optimal solutions to these problems require *expensive* non-linear mixed-integer optimizations. *Instead, we opt for heuristic solutions that provide very-close to optimal solutions.*

Our heuristic solution relies on the following subpath property: *the resource consumption, $res(\rho)$, of a path ρ can be written in terms of the resource consumption of its tail as shown in the equation in Figure 9.* In this figure, as also depicted earlier in Figure 8, $\rho'' = \rho_{1:n}$ is the (tail) path from v_1 to v_n.

We can interpret the equality in Figure 9 as follows: if ρ is the smallest resource path from v_0 to v_n, then we have

$$\alpha_{0:1} + \sigma_{0:0} \times A_{\rho''} + B_{\rho''} \leq \alpha_{0:1} + \sigma_{0:0} \times A + B$$

for all possible A and B values. In fact, we can generalize this observation to any suffix $\rho_{k:n}$ of ρ

$$\forall_{A,B} \quad B_{\rho_{k:n}} - B \leq \sigma_{0:k-1} \times (A - A_{\rho_{k:n}})$$

Low resource routing algorithm in chain networks with $\sigma \leq 1$.

- **Input:** A directed acyclic graph $G(V, E, l)$ corresponding to ABNs of a chain , $\mathcal{C}$
- **Output:** A source vertex v_0 and a sink vertex v_n

1. $G(V^t, E^t, l^t) = reverse\_topologicsort(G)$
2. For all $v_i \in V^t$, do $\sigma[v_i] = w_{s,i} + w_{s,fx,i}$
3. $parent[v_n] = \perp$
4. $A[v_n, v_n] = 1$; $B[v_n, v_n] = 1$
5. For all $v_i \in V^t$ in the topological order do
   (a) For all $e_l \in E^t$ such that $e_l = v_i \rightarrow v_j$ do
      i. $A[v_i, v_n] = \infty$; $B[v_i, v_n] = \infty$
      ii. $tempA = w_{s,i} \times A[v_j, v_n] + (w_{r,out,i} + w_{r,in,j}) \times w_{s,i}$
      iii. $tempB = B[v_j, v_n] + w_{s,fx,i} \times A[v_j, v_n] + (w_{r,out,i} + w_{r,in,j}) \times w_{s,fx,i} + w_{r,fx,i}$
      iv. if $tempA + tempB < A[v_i, v_n] + B[v_i, v_n]$ then
         A. $A[v_i, v_n] = tempA$; $B[v_i, v_n] = tempB$;
         B. $parent[v_i] = v_j$

Figure 10. Lowe res. routing in chains, case I

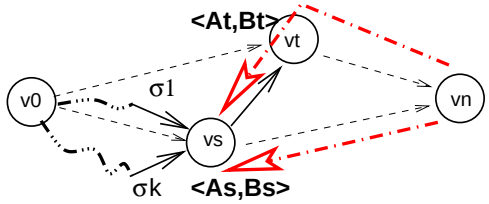


Figure 11. Edge relaxation for case II

where $\sigma_{0:k-1} = size(\rho_{0:k-1})$. On the other hand, for $\rho_{k:n}$ to be an optimal subpath, we need

$$\forall_{A,B} \quad B_{\rho_{k:n}} - B \leq 1 \times (A - A_{\rho_{k:n}})$$

Consequently, as long as $\sigma_{0:k-1} \leq 1$ (that is, if the sizes of the objects in the flow do not increase), $\rho_{k:n}$ has to be the smallest resource path.

Case I: For all v_i, $\sigma_i \leq 1$ If each node, v_i, has the property $\sigma_i = w_{s,i} + w_{s,fx,i} \leq 1$, then we can solve the smallest resource routing problem using a greedy algorithm which starts from v_n and works its way backwards to v_0 (Figure 10).

Case II: For some v_i, $\sigma_i > 1$ In this case, since $size(\rho 0 : k-1)$ is larger than 1, we can not use the above observation to find an optimal solution. We develop a Bellman-Ford style algorithm that uses edge relaxations to find a *low (but not the lowest) resource path*. The algorithm starts from node v_n and works backwards, relaxing edges, towards v_0. Consider the situation shown in Figure 11. In this figure, we have intermediary results, $\langle A_t, B_t \rangle$ and $\langle A_s, B_s \rangle$ pairs, for nodes v_t and v_s and we are considering whether to use the edge $e = v_s \rightarrow v_t$ to update the value of the $\langle A_s, B_s \rangle$ pair. There are two possibilities:

- edge e does not satisfy the relaxation condition and the value of $\langle A_s, B_s \rangle$ pair should not be updated; or

Low resource routing algorithm in chain networks with $\sigma > 1$.

- **Input:** A directed acyclic graph $G(V, E, l)$ corresponding to ABNs of a chain, $\mathcal{C}$,
- **Output:** A source vertex v_0 and a sink vertex v_n

1. For all $v_i \in V$, $\sigma\_min[v_i] = smallestsize(v_0, v_i)$ ..... `First phase`
2. For all $v_i \in V$, $\sigma\_max[v_i] = largestestsize(v_0, v_i)$
3. $numEdges = |E|$; $numVertices = |V|$ ..... `Second phase`
4. For all $v_j \in V$
   (a) $Amin[v_j, v_n] = Amax[v_j, v_n] = Bmin[v_j, v_n] = Bmax[v_j, v_n] = \infty$
   (b) $nextmin[v_j] = nextmax[v_j] = \perp$; $touched[v_j] = false$;
5. $Amin[v_n, v_n] = Amax[v_n, v_n] = Bmin[v_n, v_n] = Bmax[v_n, v_n] = 0$
6. $nextmin[v_n] = nextmax[v_n] = \perp$; $touched[v_n] = true$
7. For $count = 1$ to $numEdges$ do
   (a) For all $v_j \in V$ such that $touched[v_j] = true$ do
      i. For all $e_k = v_i \rightarrow v_j$
         A. if $touched[v_i] == false$ then $touched[v_i] = true$
         B. $tempAmin = w_{s,i} \times Amin[v_j, v_n] + (w_{r,out,i} + w_{r,in,j}) \times w_{s,i}$
         C. $tempBmin = Bmin[v_j, v_n] + w_{s,fx,i} \times Amin[v_j, v_n] + (w_{r,out,i} + w_{r,in,j}) \times w_{s,fx,i} + w_{r,fx,i}$
         D. if $Bmin[v_i, v_n] - tempBmin > \sigma\_min[v_i] \times (tempAmin - Amin[v_i, v_n])$ then
            - $Amin[v_i, v_n] = tempAmin$; $Bmin[v_i, v_n] = tempBmin$;
            - $nextmin[v_i] = v_j$
         E. $tempAmax = w_{s,i} \times Amax[v_j, v_n] + (w_{r,out,i} + w_{r,in,j}) \times w_{s,i}$
         F. $tempBmax = Bmax[v_j, v_n] + w_{s,fx,i} \times Amax[v_j, v_n] + (w_{r,out,i} + w_{r,in,j}) \times w_{s,fx,i} + w_{r,fx,i}$
         G. if $Bmax[v_i, v_n] - tempBmax > \sigma\_max[v_i] \times (tempAmax - Amax[v_i, v_n])$ then
            - $Amax[v_i, v_n] = tempAmax$; $Bmax[v_i, v_n] = tempBmax$;
            - $nextmax[v_i] = v_j$

   ..... `Third phase`
8. if $(Amin[v_0, v_n] + Bmin[v_0, v_n] \leq Amax[v_0, v_n] + Bmax[v_0, v_n])$
   (a) $next[v_0] = nextmin[v_0]$ else $next[v_0] = nextmax[v_0]$
9. $currentsize = w_{s,0}$; $tempv = next[v_0]$
10. while $(tempv \neq v_n)$ do
   (a) if $currentsize - \sigma\_min[tempv] \leq \sigma\_max[tempv] - currentsize$
      i. $next[tempv] = nextmin[tempv]$
   (b) else $next[tempv] = nextmax[tempv]$
   (c) $currentsize = currentsize \times w_{s,tempv}$
   (d) $tempv = next[tempv]$

Figure 12. Low res. routing in chains, case II

- edge e satisfies the relaxation condition and the value of $\langle A_s, B_s \rangle$ should be recalculated based on $\langle A_t, B_t \rangle$ and the parameters of the edge e.

Based on the earlier discussion, we know that the existing path from v_s to v_n is optimal if and only if

$$\forall_{A,B} \quad B_s - B \leq \sigma_{0:s-1} \times (A - A_s)$$

At this point, however, since the optimization is not completed, we do not know how the optimal path will

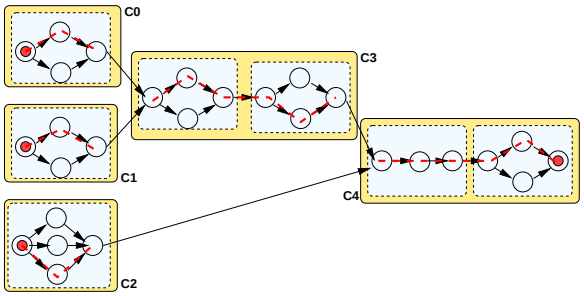


Figure 13. ABNs of a tree network

reach from v_0 to v_{s-1} and hence we do not know the value of $\sigma_{0:s-1}$. Therefore we do not have all the information required for actually checking if the edge relaxation condition holds or not. Consequently, if a node is accessible through k paths, we need to maintain k many $\langle A, B\rangle$ pairs (one for each possible path) and update each pair with each edge relaxation. This however may necessitate exponential (in the length of the chain) computation and storage. In order to prevent this exponential growth, our heuristic uses lower- and upper-bounds on the object sizes reaching the nodes in the network (*First Phase* in Figure 12). Let us denote the maximum object size reaching the node v_s as $\sigma_{s,max}$ and the minimum object size as $\sigma_{s,min}$. Both of these values can be found efficiently using the algorithm presented in Section 3.1.2.

For each node on the network, we maintain two $\langle A, B\rangle$ pairs ($\langle A_{min}, B_{min}\rangle$ and $\langle A_{max}, B_{max}\rangle$), corresponding to the lower-bound and the upper-bound of the object size. We propagate these pairs, through edge relaxation, backward from the actuator node to the source (*Second Phase* in Figure 12).

Once the source node is reached, we track our steps back (using the appropriate relaxation edges) towards the actuator, choosing the appropriate relaxation edge at each node and revealing the result path on the way (*Third Phase* in Figure 12). When we are considering node v_s on this last phase, we already have two candidate edges that we maintained (one for $\sigma_{s,min}$ and one for $\sigma_{s,max}$). However, in reality, the resource optimal path may be passing through a third edge leaving v_s. Therefore, in order to minimize the error, we compute

$$\Delta 1 = \sigma_{s,act} - \sigma_{s,min} \quad \text{and} \quad \Delta 2 = \sigma_{s,max} - \sigma_{s,act}$$

where $\sigma_{s,act}$ is the actual object sizes that are observed on this final pass towards the actuator and we choose the alternative which gives a smaller error (Δ) value to follow. The complexity of this algorithm is $O(VE)$.

3.2 Routing in Adaptive Tree and Graph Networks

In this section, we extend routing algorithms to tree networks where multiple sensors contribute to the flow of objects that reach an actuator. We use $\mathcal{T}$ to denote tree networks. It is convenient to represent the ABNs of a tree network, $\mathcal{T}$, in the form of a number of chains, $\mathcal{C}_i$ that are connected to each other in an hierarchical fashion (Figure 13). Consequently, a flow, τ, on a given tree network, $\mathcal{T}$, is a tree that is composed of paths (ρ_is) contributed by the individual chains ($\mathcal{C}_i$s) on the network. Various routing problems then can be posed as choosing a path for each chain and to create a flow tree, which satisfies certain optimality criteria.

In this paper, we do not focus on adaptive routing in tree and graph networks. Future work includes extension of the proposed routing techniques to arbitrarily complex (tree, DAG, cyclic DG) flow networks and non-linear operator semantics.

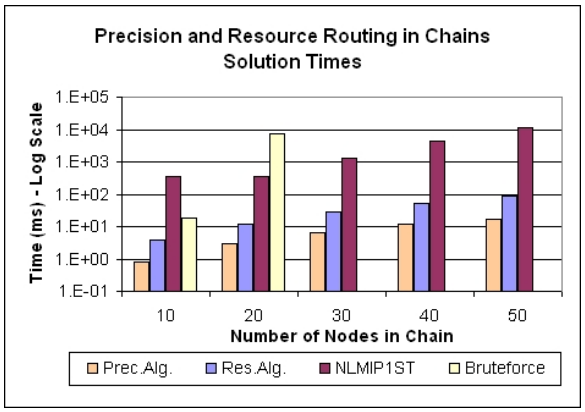
4 Experiments

For the experiments, we created a data set corresponding to various *qStream* networks. We varied the *qStream* network properties as follows:

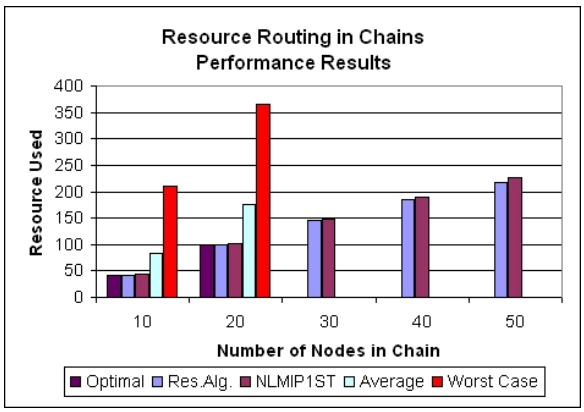
Experimental Network Properties	
Chain length	2...50 nodes
Number of behaviors per node	2...4
Precision weight	0.5...1.0
Size weight	0.2...2.0
Resource weight	1.0...10.0
Delay weight	1.0...10.0

In order to observe and interpret the performance of the proposed optimization mechanisms, in addition to our heuristics, we used two alternative optimizations: *non-linear mixed integer formulation of the problems* (NLMIP) optimized using LINGO 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.

The bruteforce and heuristic algorithms were written in Java and ran on Redhat 7.2 Linux workstations, with 1 GB RAM, 1.8 GHz Pentium IV processor, and 20 GB hard disk. LINGO Extended 8.0 with Nonlinear Global Solver was installed on a Windows PC, with 256 MB RAM, 2.0 GHz Pentium IV processor, and 40 GB hard disk. Obviously the bruteforce approach did not scale beyond embarrassingly small networks. For larger networks, 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. *Therefore, while generating the NLMIP solutions for comparison, we settled for the first solution (most of the time a local optimum) found by LINGO.* This took anywhere from 30 minutes to many hours to execute, still unusable in real settings, but significantly shorter than bruteforce optimization as well as finding global optimal of NLMIP problems. As



(a)



(b)

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 for the average fanout, and L for the chain length

Figure 14. (a) Chain optimization algorithms work many orders faster than finding the first local optima for the NLMIP solutions and (b) our results are very close to the optimal; Bruteforce results are omitted where executions took more than a day

the following results show, the *qStream* algorithms are multiple orders faster than the bruteforce and NLMIP optimizations. Furthermore, *qStream* results are within 1-2% of the optimal and most of the time better than the first local optima found by LINGO.

Precision Routing in Chains: The first observation is that the proposed precision routing algorithms indeed returned optimal results in chain networks. The average execution time of routing in chains was less than 20ms for 50 nodes per chain and an average of 4 behaviors per node (Figure 14(a)).

Resource Routing in Chains: As expected the resource routing algorithm was sub-optimal. However, the results returned by our algorithm were always within 1.0% of the optimal as shown in Figure 14(b). The average optimization time varied between 4ms for a chain of length 10 nodes to around 85ms for extremely long chains of length 50. Again, purely for comparison the bruteforce execution time for this configuration was around 17ms for chains of length 10, but quickly jumped to 7seconds for chains of length 20 (Figure 14(a)). Although not as bad as the bruteforce, the first local optima of LINGO was many orders slower than our heuristic and its results were not as good as ours (note that NLIMP1ST in the figure stands for the 1^{st} local optima for the NLIMP).

Summary: The experiments, as depicted in Figure 14, showed that the proposed heuristic algorithms indeed work efficiently (under 100ms) and effectively (within $1-2\%$ of 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 adaptive systems.

5 Related Work

Telegraph [1] provides an adaptive dataflow engine that moves data through a number of database (such as selection and join) operators. It recognizes that cost of the operators, their selectivities, and the rates at which tuples arrive from the input vary during the processing of queries. Note that most works on adaptive data operators [7, 8, 10] concentrate on data in traditional (tuple or XML) form. Therefore, they do not assume rich filtering and fusion operators. They also do not model data quality or provide explicit QoS guarantees. Although it does not address filtering and fusion operators, Aurora [3, 4, 5] focuses on QoS- and memory-aware operator scheduling and load shedding for coping with transient spikes in data rates.

There has been little prior work in using QoS in the context of data-flow architectures. Variations of the high-precision flow routing problem can be posed as a multi-constrained optimal routing (MCOR) problem, where the aim is to find a route from the source to the destination which maximizes the precision subject to delay, resource, frequency, and cost constraints. Various forms of MCOR have been studied in the literature [11, 12, 13, 14] and it has been shown that generally this class of problems are NP-complete. The two-criteria decision problem is known to be NP-complete. In [15, 16, 17], 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 *qStream* where depending on the properties of an incoming object an operator may take different actions.

On the other hand, in this paper, we *did not* address explicit multi-criteria optimization problems; but, we focussed on implicit dependencies between resource consumption, delay, and object sizes.

Q-RAM provides a set of QoS optimization schemes, with discrete QoS options, in the context of videoconferencing [19, 20]. Unlike *qStream* , which considers workflows, Q-RAM considers each task as an independent application and imposes the quantification of the system utility on users via weighting the linear combination of all quality dimensions of all applications. [21] proposes a QoS Broker which can orchestrate resources needed by various applications. The broker uses profiles to find out the resource requirements and QoS characteristics of the tasks at the application and transport layers and negotiates with the corresponding resource managers for the resources required by these applications. [22] proposes a negotiation and resource reservation protocol (NRP) for multimedia applications. NRP allows applications to specify ranges of acceptable QoS parameters. After mapping application level QoS specifications to network level QoS, NRP calculates the acceptable resource ranges for each link and performs negotiations. [23] proposes a multimedia application protocol, MMAP, which, given an orchestrated multimedia document, computes corresponding QoS parameters (using a traffic model, based on buffer availabilities) and negotiates with the network service providers. In [24], we introduced scheduling techniques that benefit from flexibilities in the temporal specifications of multimedia presentations for dealing with the buffer and network resource constraints. Unlike most of these works which consider brokering of resources for *independent* applications, *qStream* focusses on QoS-based *workflow management* in a data streaming environment.

6 Conclusions

We presented a novel quality-adaptive real-time flow architecture (*qStream*). *qStream* operators filter various features from streamed data, and fuse and map media streams onto output devices as *constrained by the QoS*. The nodes in the flow network correspond to quality-adaptive computing elements. Here, we focussed on this quality-adaptive nature of *qStream* and presented algorithms that efficiently identify flow plans that delivers high precision, small delay, and small resource object streams in *chain-structured* networks. Future work includes extension of the proposed routing techniques to arbitrarily complex (tree, DAG, cyclic DG) flow networks and non-linear operator semantics. We are also developing runtime correction techniques which rely on the presented heuristics. These complement the optimizer by modifying plans as needed during the operation of *qStream* .

References

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

[2] F.Tian and D.DeWitt, *Tuple Routing Strategies for Distributed Eddies,* VLDB 2003.

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

[4] D.Carney *et.al. Operator Scheduling in a Data Stream Manager.* VLDB 2003.

[5] N. Tatbul *et. al. Load Shedding in a Data Stream Manager*, VLDB 2003.

[6] D.J.DeWitt *et al. Gamma - A High Performance Dataflow Database Machine.* VLDB 1986.

[7] J.M.Hellerstein *et al. Adaptive Query Processing: Technology in Evolution.* IEEE Data Eng. Bltn. 23(2): 7-18, 2000.

[8] S.Madden and M.J.Franklin. *Fjording the Stream: An Architecture for Queries over Streaming Sensor Data,* ICDE 2002.

[9] S.Babu, *et al. Adaptive Ordering of Pipelined Stream Filters.* SIGMOD 2004.

[10] S.Babu and J.Widom. *Continuous Queries over Data Streams.* SIGMOD Record, Sept 2001.

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

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

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

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

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

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

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

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

[19] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek, *On Quality of Service Optimization with Discrete QoS Options,* IEEE Real Time Technology and Applications Symp., 1999: 276-.

[20] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, *A Scalable Solution to the Multi-Resource QoS Problem,* IEEE Real-Time Systems Symp. 1999: 315-326.

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

[22] G. Dermler *et al. A Negotiation and Resource Reservation Protocol (NRP) for Configurable Multimedia Applications.* ICMCS 1996: 113-117

[23] S.V. Raghavan, B. Prabhakaran, and S.K. Tripathi, *Handling QoS Negotiations in Distributed Orchestrated Presentations*, Journal of High Speed Networking, Vol.5, No.3, pp 277-292, 1996.

[24] K.S. Candan, B. Prabhakaran, V.S.Subrahmanian. *Retrieval Schedules Based on Resource Availability and Flexible Presentation Specifications* Multimedia Syst. 6(4): 232-250, 1998.