GAL: A Middleware Library for Multidimensional Adaptation

DAVID GOTZ and KETAN MAYER-PATEL
University of North Carolina at Chapel Hill

Data adaptation is an essential system component in a wide variety of application areas. Adaptation is performed to manage data in response to limited resources and changing system conditions. Recent research has led to the development of general models for adaptive systems, including our own general framework for multidimensional adaptation. In this article, we review our framework, which distills the common elements essential to a broad class of adaptive applications. We then present our design for GAL, a middleware library which implements our generic framework. We include a thorough evaluation of GAL's adaptive performance in our experimental prototype.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.1.1 [**Information Storage and Retrieval**]: Systems and Information Theory—*Value of Information*; I.3.2 [**Computer Graphics**]: Graphics Systems—*Distributed/Network Graphics*

General Terms: Algorithms, Performance, Measurement

Additional Key Words and Phrases: Adaptation, Multimedia

1. INTRODUCTION

Advances in data capture, storage, and processing technologies now allow scientists to collect, simulate, or create immense collections of data. Unfortunately, advances in networking technologies, while impressive, have not kept pace. This has led to a growing gap between the amount of data we can capture, store, and process, and the resources we have to transmit that data over the network. Because of this performance gap, adaptation has become increasingly central to the performance of high-bandwidth applications such as remote data visualization, tele-immersion, and media streaming.

To date, adaptation techniques, algorithms, and frameworks have largely been developed within the context of a specific application or data type. For example, Geographic Information Systems (GIS) make extensive use of multiresolutional data for visualization of terrain information [FM02]. In graphics, multiresolutional geometry information is used to create hierarchical levels of detail (HLODs) [Lue01]. These are used to dynamically adjust model complexity in order to achieve a target rendering rate. In multimedia, layered media encodings are used to dynamically scale media bitrates to match current network conditions [RHE99].

The challenge we face when tackling the adaptation problem in whatever con-

Author's address: D. Gotz, Department of Computer Science, UNC-Chapel Hill, CB #3157 Sitterson Hall, Chapel Hill, NC 27599

text it appears is twofold. First, how can we compactly and intuitively specify adaptation policy to support specific user-level goals? Second, given a particular adaptation policy and set of user-level goals, how can we efficiently evaluate that policy relative to available resources and the way the data are represented and organized?

While ad hoc application-specific methods may be effective when the number of dimensions for adaptation is limited, the problem can become overwhelming as the number of dimensions increases. The same is true when an adaptation decision must negotiate between data sources of fundamentally divergent natures. The complexity of these mechanisms may become very difficult to correctly manage. Expressing adaptation policy in a rule-based manner, for example, becomes painful as the number of possible tradeoffs grows.

In our earlier work [GMP04], we proposed a conceptual framework for general multidimensional adaptation. The framework expresses adaptation as a maximization problem using a spatial utility metric. We defined a generic graph-based data representation that can express both syntactic and semantic data relationships. We then formalized a set of adaptation operations which act upon the representation graph.

In this paper, we discuss the design and evaluation of a middleware library based on our framework. We call our library *GAL: A Generic Adaptation Library.* we review the underlying conceptual framework, discuss our library design, and evaluate the library as part of a image-based model streaming application prototype. Our evaluation shows GAL to be effective at adapting the client prototype's data stream in the face of a number of dynamic application conditions.

1.1 Organization

The remainder of this article is organized in the following manner. In Section 2, we present background information and review our own related work. We discuss the design of our library in Section 3. We describe our metric for evaluating the effectiveness of our adaptation library in Section 4. We present our performance evaluation in Section 5. Finally, we discuss future work and conclude our paper in Section 6.

2. BACKGROUND

There has been a vast amount of research exploring adaptation mechanisms for networked applications. As we stated in the introduction, the solutions are typically developed in an ad hoc manner to support only a specific application. However, a handful of research groups have explored the adaptation problem more widely and developed more general models for adaptive systems. In this section, we present a brief overview of these other models, followed by a more detailed description of our own model which forms the basis of the work in this article.

2.1 Generalized Adaptation

Several research groups have proposed generic models for adaptive systems. Jonathan Walpole et al. proposed *Quality-of-Service Semantics* [WKL+99] for the use in a range of multimedia systems. In this work, a multidimensioal presentation state space is defined where each dimension corresponds to a vector of adaptive control.

They define a single point within that space as the ideal state, and measure quality-lost as a distance between that ideal point and the current presentation state. The quality-loss value is used to drive adaptive changes to the presentation state.

In other work, Bowers et al. proposed the use of *Adaptation Spaces* [BDM+00] to model adaptive behavior. In this technique, individual implementation alternatives are modeled as nodes within a graph. Edges within the graph correspond to valid state transitions. Transitions are guided by an application-specific quality value assigned to each node.

Chatterjee et al. propose a multidimensional *Benefit Function* [CSSL97]. The function is evaluated over a space defined by dimensions that correspond to degrees of freedom within the adaptive application. The static benefit function is predefined and is used to guide tradeoffs between adaptive dimensions.

2.2 Our Model for Multidimensional Adaptation

This subsection outlines our general framework for adaptation [GMP04]. We develop a graph-based data representation abstraction, which is embedded within a multidimensional utility space. We then pose the task of adaptation as a maximization problem.

2.2.1 *Illustrative Example.* Throughout this section, we illustrate our framework via a simplified sample application. For each new concept, we present a formal definition followed by a concrete example. The sample application is a simplified computer graphics system where a user navigates through a one dimensional scene composed of a collection of geometric objects, each of which is stored at multiple resolutions. We assume that the geometric models are layered in the sense that lower resolution models are required to decode higher resolution information. At runtime, the example application should adapt the flow of incoming data to reflect both limited rendering resources and a moving viewpoint within the one dimensional scene.

2.2.2 *Representation Abstraction.* Our graph-based representation abstraction is embedded within a multidimensional *utility space*. Each dimension of the utility space corresponds to a degree of freedom in the adaptive application. For example, a video streaming system that allows adaptation over both frame rate and image resolution might have a two-dimensional utility space. For multimedia applications, each media object is embedded within an independent *media subspace*.

For example, our sample application has a two-dimensional utility space . One dimension defines the spatial position of geometric objects within the scene, while the second second dimension is used to reflect the resolution level of each representation of an object. Because our sample application consists of only a single media type, there is just one media subspace defined as equal to the overall utility space. We illustrate the utility space in Figure 1(a).

In our abstraction, nodes map to individual elements and represent an atomic unit of information. Each node n_i is associated with a single media object and is located at a specific point $\mathbf{Pos}\{n_i\}$ within the utility space. The relative positions between nodes are used to express the semantic relationship between individual elements of information.

In our running example, we use nodes to represent the model of an individual

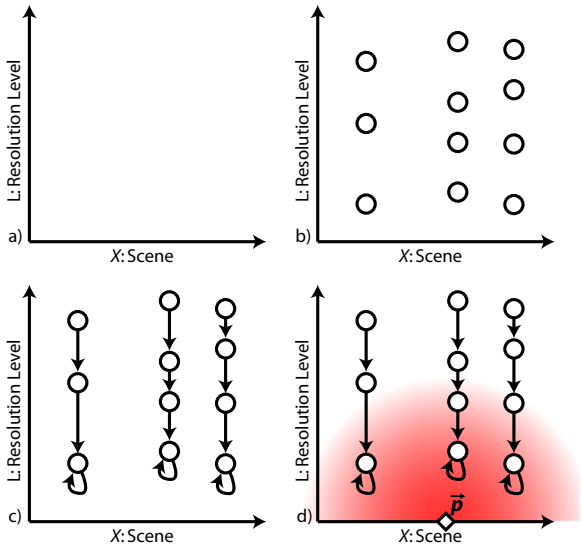

Fig. 1. (a) The sample application's utility space is defined by the spatial position within the scene, X, and the resolution level, L. (b) Nodes are used to represent individual geometric models and are embedded within the utility space according to their resolution and position. (c) Edges are used to illustrate the data dependence between models that represent the same object at different resolution. (d) A point of interest, $\vec{p}$, is maintained within the utility space. The utility of a given node can be measured as a function of the distance between the node and $\vec{p}$ as illustrated by the shaded region in this figure.

geometric object at a specific resolution. The position of each node within the utility space is determined by both the object's spatial position within the scene and its resolution. In Figure 1(b), we illustrate three geometric objects at different positions in the scene.

Each node is assigned one of four states. This state, $\mathbf{State}\{n_i\}$, can change over time through state transitions. In addition, specific applications may define auxiliary meta-data values for each node. These values can be used to represent more specialized application information.

Data dependencies between nodes are represented by directed edges. An edge e_i has both a source node $\mathbf{Src}\{e_i\}$ and a destination node $\mathbf{Dest}\{e_i\}$. An edge in our abstraction both expresses data dependence and corresponds to the specific bytes of information needed to resolve $\mathbf{Src}\{e_i\}$ given that we have already resolved $\mathbf{Dest}\{e_i\}$. An edge therefore expresses the syntactic relationship between a pair of nodes. The data required to resolve a node without any prior knowledge is expressed via a *self-edge*. A self-edge is defined as an edge e_i where $\mathbf{Src}\{e_i\} = \mathbf{Dest}\{e_i\}$.

In our example application, edges represent dependencies between different resolution models of the same geometric object, as shown in Figure 1(c). The model with the greatest error is fully encoded without any data dependencies. This leads to the inclusion of a self-edge for nodes representing an object at the coarsest level. The predictive relationships between nodes are expressed through directed edges pointing from higher resolution nodes to lower resolution nodes.

Furthermore, we define a *cluster* as a group of one or more edges. Each cluster c_i contains a list of all edges assigned to it. In addition, a cluster maintains a cost estimate which measures the cost of loading all edges assigned to the cluster. When performing load operations, the data associated with all edges in a cluster is treated as an atomic unit. Adaptation of the datastream is therefore performed at cluster-level granularity.

In our sample application, we need to load each resolution of each object independently. We therefore assign every edge to its own unique cluster.

2.2.3 *Point of Interest and Prediction Vector.* An adaptive application must maintain a *point of interest* which moves within the utility space. This point is part of a larger *prediction vector* which contains both the current point of interest and zero or more predictions of future interest points. Each vector entry pairs a point in the utility space with a confidence value. The prediction vector is used to represent both current system preferences and predicted future needs.

Our simplified example would use a prediction vector of length one, containing only the point of interest. The position of the point of interest within the utility space is determined by the user's current position within the scene. The prediction vector is illustrated in Figure 1(d) as a diamond marker.

2.2.4 *State Transitions.* Each node in our representation is assigned to one of four possible states. A node's state may change over time, but it has just a single state at any particular moment in time. A node's state reflects the current status of the information represented by that node. The possible states are as follows:

—**Idle**: The information for this node is not resolved. Nor is it possible to resolve without resolving some other node first.

—**Available**: The information for this node is not resolved. However, it is possible to resolve without resolving some other node first.

—**Active**: The information for this node is in the process of being resolved.

—**Resolved**: The information for this node is resolved.

The state of each node can change over time by either *promotion* to a higher state or *demotion* to a lower state. In addition, after each transition, a number of state invariants must be reinforced. These invariants are discussed in detail in [GMP04].

2.2.5 *Supporting Utility and Cost Evaluations.* The notions of *utility* and *cost* are fundamental concepts essential to adaptation. Individual elements of information are either more or less useful to an application than others. We define this notion as the utility of information. At the same time, access to a unit of data comes at some cost, often measured in time or required resources.

Given a set of available elements, the process of adaptation attempts to maximize the utility of the received information and minimize the associated cost. When noted as a ratio of utility with respect to cost, adaptation becomes an attempt to maximize the *utility-cost ratio*, or UCR.

Using the UCR, we frame the task of adaptation as a maximization problem where the application computes a utility measure and a cost measure for a set of possible adaptive options and chooses the option with the highest UCR value.

Variable	Description
$\mathbf{U}$	Utility Space
$\mathbf{M}_i \subset \mathbf{U}$	Media Subspaces
$\mathbf{N} \subset \mathbf{U}$	Navigable Subspace
S	Set of all nodes
n_i	A node from the set S
$\mathbf{Pos}\{n_i\}$	The position of n_i
$\mathbf{State}\{n_i\}$	The state of n_i
$\mathbf{Arr}\{n_i\}$	The list of arriving edges at n_i
$\mathbf{Dep}\{n_i\}$	The list of departing edges at n_i
B	The set of base nodes (nodes with a self-edge)
A	The availability front (all nodes in state *Available*)
E	Set of a edges
e_i	An edge from the set E
$\mathbf{Src}\{e_i\}$	The source node for e_i
$\mathbf{Dest}\{e_i\}$	The destination node for e_i
$\mathbf{Clust}\{e_i\}$	The cluster to which e_i belongs
C	The set of all clusters
c_i	A cluster from the set C
$\mathbf{Edges}\{c_i\}$	The list of edges in c_i
$\mathbf{Cost}\{c_i\}$	The cost estimate for c_i
$\vec{p}$	The prediction vector
$\vec{p}[i]$	The ith element of $\vec{p}$
$\mathbf{Pos}\{\vec{p}[i]\}$	The position of $\vec{p}[i]$
$\mathbf{Con}\{\vec{p}[i]\}$	The confidence value for $\vec{p}[i]$
α	The set of scale dimensional factors
$\alpha_i \in \alpha$	An individual scale factor

Table I. The parameters of our representation graph model.

Both the utility of information and the cost of acquiring it are application specific properties. Our framework provides general tools to make these evaluations but leaves the formulation of specific metrics to the application designer. To evaluate the UCR, an application must first define two metrics. The first metric measures the utility of an individual node. The second metric determines the cost of resolving that node.

First, we define an abstract utility metric, $UtilMetric$, used to evaluate the usefulness of each node $n_i \in A$. $UtilMetric$ evaluates the utility of a single node as a function of the node itself, the overall utility space, the set of all nodes, and the prediction vector. The implementation of this metric must be defined by the application to meet system-specific needs.

For example, in our sample application we need a metric that reflects our need for both low-error and nearby models. This can be achieved by computing the inverse of the distance between the prediction vector and a node. Using this metric, nodes closer to the point of interest are assigned a higher utility.

Second, we define an abstract cost metric, $CostMetric$, used to evaluate the minimum cost of resolving a node n_i. The cost metric in our running example is simply the number of bytes required to resolved the node in question.

Finally, we define the utility-cost ratio as shown in Equation 1.

$$UCR(n_i, \mathbf{U}, S, \vec{p}) = \frac{UtilMetric(n_i, \mathbf{U}, S, \vec{p})}{CostMetric(n_i)} \quad (1)$$

2.2.6 *Managing Dimensional Tradeoffs.* Our representation abstraction maintains a set of nodes located within a multidimensional utility space. We then express the utility of an individual node as a geometric function within that space. We can therefore manage the tradeoffs between dimensions in our utility space by performing scaling operations on individual dimensions. We define the set of scale factors as $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$, where n is the number of dimensionality of the utility space. We call this the *alpha vector*.

Geometrically scaling the utility space in a specific dimension will result in biasing the $UtilMetric$ evaluation and alter the importance of a specific dimension. An application can adjust the relative importance between dimensions by changing values in the alpha vector.

A full table of the variables in our adaptation model is shown in Figure I.

3. LAYERED SYSTEM DESIGN

We have developed a layered system design for the implementation of our multidimensional adaptation framework. This design considers a general client-server application model where adaptation is performed on the client, which in turn drives requests for additional data to be delivered by the server.

Given this system architecture, we introduce an extra system layer between the application and communication layers. This middle layer performs adaptation with input from neighboring layers in order to incorporate application and network measurements into the adaptive algorithm. In this section, we discuss our layered design and briefly outline the interface between each layer.

3.1 Three Primary Layers

Our design consists of three primary layers: (1) the application layer, (2) the adaptation layer, and (3) the communication layer. A single client application is composed of the union of these three layers. Our middleware library, GAL, is the middle adaptation layer. The middleware library sits between the core networking functionality of the network protocol-specific communication layer and the application-specific application layer. The layered design is illustrated in Figure 2.

During normal operation, application specific requirements, such as the point of interest, alpha vectors, and other user preferences are maintained by the application. The adaptation layer can request these values from the application layer as needed. Similarly, network measurements including loss and latency estimates can be requested from the adaptation layer. These values are typically used in cost and utility evaluation within the adaptation layer. Once the adaptation layer chooses a node for resolution, the request is passed down to the communication layer which then relays the request to the server via the Internet. The specific mechanism for these requests depends upon the specific network protocols being used. However, we define an interface for such requests as described in Section 3.2.

Once data has been received by the communication layer as a series of packets,

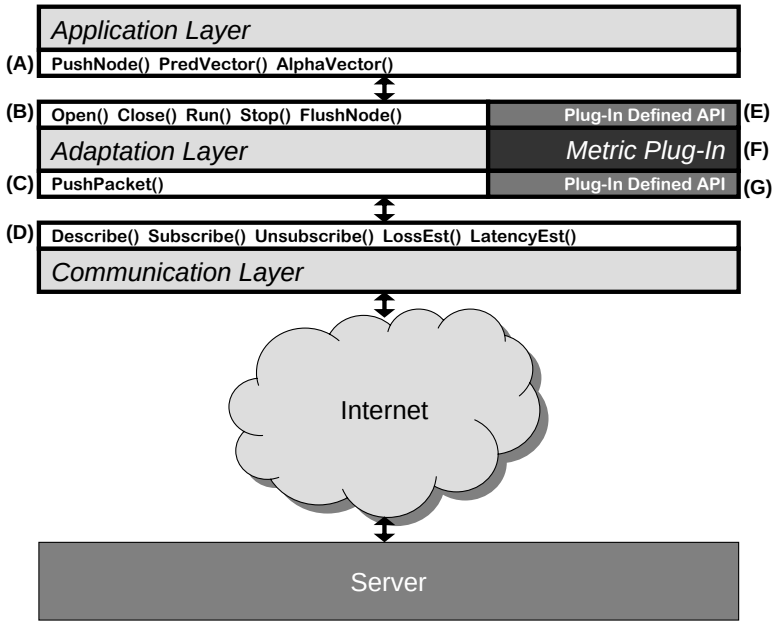

Fig. 2. Our system model consists of three layers: the application layer, adaptation layer, and communication layer. The Generic Adaptation Library, or GAL, is an implementation of the adaptation layer. It also defines a number of application programming interfaces labeling with letters A-G in the figure. Custom cost and utility metrics are supported as plug-ins.

it is passed up to the adaptation layer where the state of the representation graph is updated to reflect the new state of the system and packets are reassembled into application data units (ADUs). Once ADUs have been reassembled and recorded within the representation graph, they are made available to the application.

3.2 Layer-to-Layer Interfaces

The GAL library provides two primary application programming interfaces (APIs). One API defines the application-to-adaptor interface. The second API defines the communicator-to-adaptor interface. In addition, GAL provides templates for both the Application and Communication layers that define a set of minimal functionality that must be supported by each layer.

3.2.1 *Application-Adaptor API.* The application-to-adaptor interface defines a set of functions used by the application to interact with the adaptor library. A simplified API is illustrated in Figure 2B. The API includes functions `Open()` and `Close()` which initiate and terminate a new session,respectively. During an ongoing session, the iterative adaptation algorithm can be controlled via the `Start()` and `Stop()` functions. These four function allow an application to perform session control and to manage the adaptation thread.

The final function in this portion of the API is `FlushNode()`. While the adaptation thread is executing, incoming data is passed to the application. However, the data does not persist on the client indefinitely. When data is no longer needed or

can no longer be stored (i.e. a limited cache size), the application must notify the adaptation logic that the information is about to be evicted. This is done via the `FlushNode()` function.

3.2.2 *Communicator-Adaptor API.* The communicator-to-adaptor interface defines a set of functions used by the communicator to interact with the adaptor library. A simplified API is illustrated in Figure 2C. This portion of the API is very simple and supports only one key operation: the arrival of data packets. The `PushPacket()` function is used as a callback function by the communicator every time a new packet arrives from the network. To allow for the use of several transport protocols, the arriving stream of packets can be missing lost packets and packets can be out of order. Reordering is performed within the adaptation layer.

3.2.3 *Application Layer Template.* The application layer is an abstraction for the application-specific code required by any actual system. As such, this layer will vary greatly for each application. However, an application must define a minimal set of functionality to work correctly with GAL. This interface, illustrated in Figure 2A, is used by GAL to exchange data and monitor changing application conditions.

The first function handles data exchange between the adaptation and application layers. As a node enters the resolved state within the adaptation layer, the data corresponding to the node is pushed to the application layer via the `PushNode()` function. An application's implementation of this function determines what happens to data once it becomes available to the application.

The remaining functions are used to provide the adaptation layer with access to the current application conditions. The `PredVector()` function returns a reference to the current prediction vector, while the `AlphaVector()` returns a reference to the current alpha vector. For more information on these constructs, see the brief overview in Section 2.2 or refer to our paper detailing the overall adaptation framework [GMP04].

3.2.4 *Communication Layer Template.* The communication layer is an abstraction for the portion of the system that handles the low level network functionality. This layer is specified to allow GAL to work equally well with a large set of transport level network protocols. GAL depends on at least best-effort packet delivery and uses a subscription metaphor for data access. Based on the template set of functions, a new communicator must be defined for whatever network protocol is used by the application.

The first function defined in this template is the `Describe()` function. This function must be defined to utilize the available network resources to obtain a description of the utility space and representation graph from the specified server. When GAL's `Open()` function is called by an application with the appropriate dataset information, a call is made by the adaptation layer to `Describe()` to retrieve the specified dataset description.

The `Subscribe()` and `Unsubscribe()` functions are used to support the subscription oriented communication model used by GAL. When the adaptor chooses to activate a specific cluster, it uses the `Subscribe()` function to initiate the data request. A corresponding call to `Unsubscribe()` terminates the request. In a typical TCP-based communicator, these functions would correspond to the setup and

tear down of a network socket connection.

Finally, two measurement functions are required so that the adaptor can effectively monitor network performance and adapt accordingly. The `LossEst()` function provides a loss estimate which can allow the adaptor to respond to congestion events. This is critical when the communicator utilizes non-responsive transport protocols such as UDP for transmission. The `LatencyEst()` must provide an estimate of the subscription latency, defined as the time between the initiation of a subscription request and the arrival of the first packet of data.

3.3 The Metric Plug-ins

Our adaptation framework allows for custom cost and utility metrics to be implemented as needed. We define the notion of a *metric plug-in* to support this feature. New metric plug-ins can be defined to implement any desired utility or cost metric. The plug-in is given access to the internals of the adaptation layer, including the representation graph. The plug-in architecture is illustrated in Figure 2F.

Standard inputs to the cost and utility metrics, such as the prediction and alpha vectors or network measurements, can be accessed through the previously presented APIs. However, some metrics may need additional information from either the application or communication layers. We therefore allow each metric plug-in to define its own API to be exposed to these two layers. This is shown in Figures 2E and 2F.

4. SYSTEM PERFORMANCE METRIC

In this section, we propose an application-independent performance metric for evaluating the performance of our adaptation layer algorithms and the GAL middleware library. Our goal is to define a performance metric at the adaptation layer this is independent of any application layer performance issues. We achieve this goal with the *Summed Utility Metric* (SUM), a metric that measures system performance as a function of the current state of the representation graph.

Our performance metric, SUM, is designed to provide a numerical measure of system performance at a given point in time, independent of any application level knowledge. We therefore formulate the SUM using only information that is available to the adaptation layer. This includes the representation graph and the utility metric.

The SUM is derived from the notion that the adaptation layer's performance can be measured by the utility of the data it has obtained at any given point in time. This can be measured by applying the current utility metric to every resolved node in the representation graph. We then sum all of the resolved node utility values to find the SUM. We formally define the SUM metric in Equation 2.

$$\mathbf{SUM} = \sum_{n_i} UtilMetric(n_i) : n_i \in S \wedge \mathbf{State}\{n_i\} = Resolved \tag{2}$$

It is important to note that the SUM is a measure of performance at a single point in time. To capture a reliable measure of system behavior, the SUM metric must be evaluated repeatedly over a period of time.

5. EXPERIMENTAL PROTOTYPE AND RESULTS

We have performed a series of experiments to examine the performance of GAL under a number of operating conditions. Using the SUM as our performance metric, our results show that GAL is an effective design for supporting adaptive applications that face dynamic operating conditions.

In this section of our article, we will first describe the target application for our experimental prototype. We will then describe our testing environment and methodology. Finally, we will present the results of several experiments.

5.1 Target Application

In our experiments, we implemented an image-based rendering (IBR) streaming application for our evaluations. This target application is a good match for our framework because it is a high-bandwidth application with several dimensions of adaptability and a dynamic point of interest.

Our prototype system is based on the Sea of Images (SOI) algorithm [AFYC02] by Aliaga et al. In this algorithm, the system takes as input a large set of high resolution cylindrical images captured at various points along a eye-level plane. This can be done, for example, with a camera attached to a motorized cart. A Delaunay Triangulation of the image positions is computed to build a triangle mesh. At run time, a user can navigate a virtual viewpoint along the eye-level plane. A virtual rendering of the scene is synthesized by interpolating between the three nearest pictures found within the input dataset, as determined by the triangle containing the viewpoint. This process is illustrated in Figure 3.

The input dataset is often too large to be loaded entirely into memory. As a result, the three closest images may not always be available. Therefore, the triangular mesh is determined only by the set of loaded images, and is dynamically maintained as images are loaded into or removed from memory.

Our experimental prototype is a SOI-based IBR streaming application, where individual clients can access a centrally stored image database and independently navigate through a recreated scene. As a motivating example, imagine a digital museum sharing a large IBR model of a famous location for virtual visitors connected to the Internet.

Our prototype is designed as a client-server system with a single image server that can transmit a stream of images to a set of interested clients. Each client is able to navigate through the space independently along their own unique paths. For this reason, each client must be able to adapt their own data flow based on their particular needs. We satisfy this design requirement by using GAL as the adaptive engine of our prototype.

The first step in developing a GAL-based design for our target application is to identify the dimensions of adaptability. These dimensions define our utility space. We designed a hierarchical and multiresolutional data structure for the images that provided five such dimensions. Our representation allows for clients to access the image data by choosing a location along the 2D plane, a view direction, image resolution, and a spatial density value. More information on our representation can be found in our earlier work on the scalable delivery of IBR datasets [GMP05].

Once the utility space has been defined, we map the data representation to a set

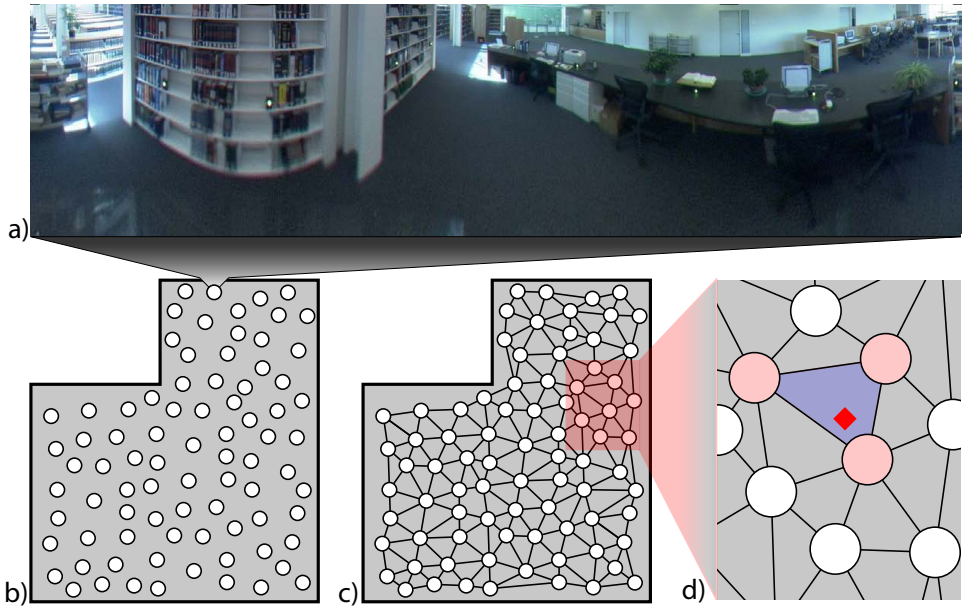

Fig. 3. The Sea of Images rendering algorithm proposed by Aliaga et al. takes as input a large set of panoramic images. Using the database of images, novel views of the scene can be constructed by interpolating between triplets of stored images. (A) Each panoramic image in the input dataset contains a full 360° view of the scene. (B) The input images are captured by a camera positioned along a plane at eye-level. (C) At runtime, a Delaunay Triangulation of the stored image positions is maintained. (D) As the user moves through the scene, illustrated by the red diamond, the triangle containing the user's position determines the appropriate triplet for interpolation.

of nodes, edges and clusters that make up our final representation graph. In this application, each node corresponds to a portion of an image from the input dataset. The position of each node is determined by the specific location of the image on the eye-level plane, the view direction, and the resolution and spatial density levels. The graph's edges represent encoding dependencies. For example, our multiresolution image representation introduces dependences between the different resolutions of each image. These dependencies are expressed by including edges between nodes corresponding to images at the same location and view direction but of different resolution. Finally, we introduce clusters to our representation graph which specify the granularity with which we are able to access the image data. In our experiments, we vary the degree of clustering to explore its impact on performance. This is discussed in more detail when we present the results in Section 5.3.

The original input dataset consists of about 2,000 cylindrical color images, each with a resolution of 2,048×512. Our experiments are all performed using a representation graph with 15,568 edges connecting 15,568 nodes. The number of clusters varied widely, ranging from 16 to 15,568.

The clustering parameter is used to control the granularity of data access. At one extreme, a data set would contain just one cluster. This would correspond to

a single large file containing all of the data. A user would have no control over the order in which data was received. At the other extreme, every edge would be assigned to a unique cluster. This would make every edge individually addressable and would providing the greatest control over the order in which data was received. In this scenario, a client application could choose to receive any of the available clusters in an arbitrary order. In our experiments, we evaluate system performance over a range of clustering parameter values.

As dictated by our design of GAL, the application layer contains the rendering code and manages the prediction and alpha vectors. The point of interest corresponds to the clients position along the eye-level plane and alpha vector values can be changed to manage the tradeoffs between image resolution, spatial density, position, and view direction. For example, a fast moving user may require low resolution images from farther away while a slow moving user would require high resolution images from very close to the point of interest.

The communication layer contains all of the lower level network code. In our experiments, this layer uses TCP/IP to obtain reliable, in-order data delivery between the client and server. TCP also includes congestion control which becomes important as the number of concurrent users grows past the amount of available bandwidth.

5.2 Testing Environment and Methodology

We performed a series of experiments using the IBR Streaming Application layer, our GAL library adaptation layer, and the TCP/IP based communication layer. All of our experiments were executed using the Emulab network emulation testbed [WLS+02]. In all of our network topologies, we provisioned a single server with a 100Mbps network connection. Our network model assumed that all bandwidth bottlenecks occur within the "last mile" for each client. We therefore modeled all core links within our topology with the same 100Mbps bandwidth as the server. Links connecting clients to the core network were given a fixed bandwidth of between 0.1Mbps and 10Mbps depending on the experiment.

For each experiment, clients navigated a ten minute path through the IBR dataset. The path included a variety of movement types including both fast and slow movements and changes of direction. During the ten minute execution time, we computed a value for the SUM metric once per second. When presenting average SUM values, we felt it was important to ensure that our results included only steady-state behavior. We therefore considered only the second five minutes of SUM data whenever average values are presented.

5.3 Results

We performed several experiments to evaluate the adaptation performance of GAL and to explore a number of engineering decisions that must be made during system implementation. We present the most interesting of these results in this subsection.

5.3.1 *Session Adaptation over Time.* For our first experiment, we examined the SUM values over the course of a single ten minute session. To smooth out the SUM data and to make it easier to visualize, we plot the cumulative total of SUM as time progresses. We show the results from three such sessions, each using

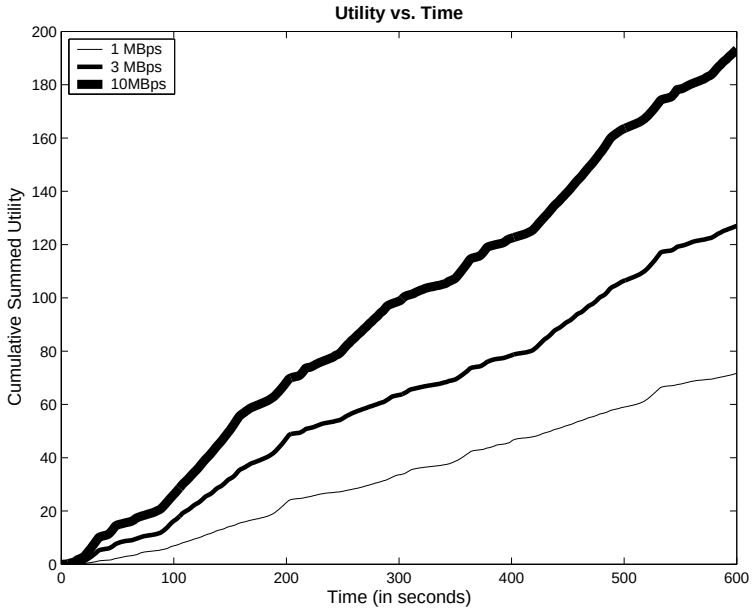

Fig. 4. This graph shows three plots of cumulative summed utility values over the course of a ten minute session. The plots correspond to three different bottleneck link speeds. The steady growth of the three plots shows that over time, GAL continues to provide data of high utility. The difference in slope shows that GAL properly utilizes any available bandwidth to improve utility as much as possible.

a different bottleneck bitrate, in Figure 4. In all of these sessions, we emulated a single active client and our representation graph was configured to place each edge in its own cluster, yielding 15,568 unique clusters.

The results indicate that the GAL layer performs well in several important ways. First, they show that despite dramatically different behavior during various time periods, the cumulative SUM value continues to grow steadily. This indicates that under all conditions, the GAL layer is able to reliably deliver useful data to the application layer, as judged by the utility metric. Second, the similar shape but disparate slope for each plot shows that GAL appropriately makes use of additional bandwidth to improve the overall utility of the set of resolved nodes.

5.3.2 *Adaptation Impact of Bottleneck Link Speeds.* In a separate experiment, we calculated the average SUM value for experiments using several different bottleneck link speeds. In all of these sessions, we emulated a single active client and our representation graph was configured to place each edge in its own cluster, yielding 15,568 unique clusters. The results are shown in Figure 5.

This experiment confirms what we learned from the first experiment: that GAL correctly takes advantage of excess bandwidth to improve the utility of the set of resolved nodes. However, it is important to notice the shape of the curve in the plot. The steepest gains in utility are found when increasing bandwidth from a very low value. The gains tend to plateau as the bottleneck link speed grows faster

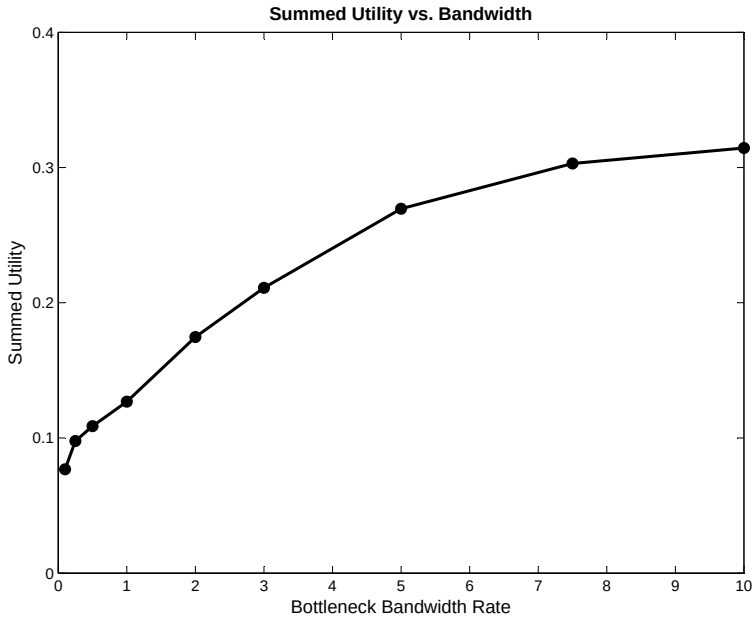

Fig. 5. This figure shows the average summed utility metric value for several sessions. Each session was executed with a different bottleneck link speed. As expected, the plot shows an increase in utility for higher link speeds. However, the shape of the curve is not linear. This highlights the diminishing marginal benefits of additional bandwidth.

and faster. While not immediately obvious, the decrease in marginal utility at higher bitrates is the expected result. Suppose that with a bottleneck bitrate x, a system using GAL is able to resolve the n nodes with the highest utility values. Meanwhile, a system with a bottleneck bitrate of $2x$ would be able to resolve those same n nodes plus n additional nodes. However, if GAL is operating correctly, the additional n nodes should never be as useful as the first n. As a result, there is a lower marginal utility for the added bandwidth.

5.3.3 *Adaptation Performance With Large User Groups.* The GAL design places the burden of adaptation on the client side. This is particularly critical when there are a large number of users simultaneously accessing the server. If the server were responsible for adaptation, the server's CPU load would grow linearly with the number of clients. However, our client-side adaptation architecture means that the server's CPU doesn't perform any adaptation tasks. Instead, the number of users is limited by the amount of outgoing bandwidth provisioned to the server. To test this property, we ran several sessions with a variety of user group sizes.

For all sessions in this experiment, clients were provisioned with a 5Mbps bottleneck link and navigated independently through the IBR dataset. Our representation graph was configured to place each edge in its own cluster, yielding 15,568 unique clusters.

The results from these sessions are shown in Figure 6. As the plot illustrates, there is essentially no drop-off in performance until the number of clients surpasses

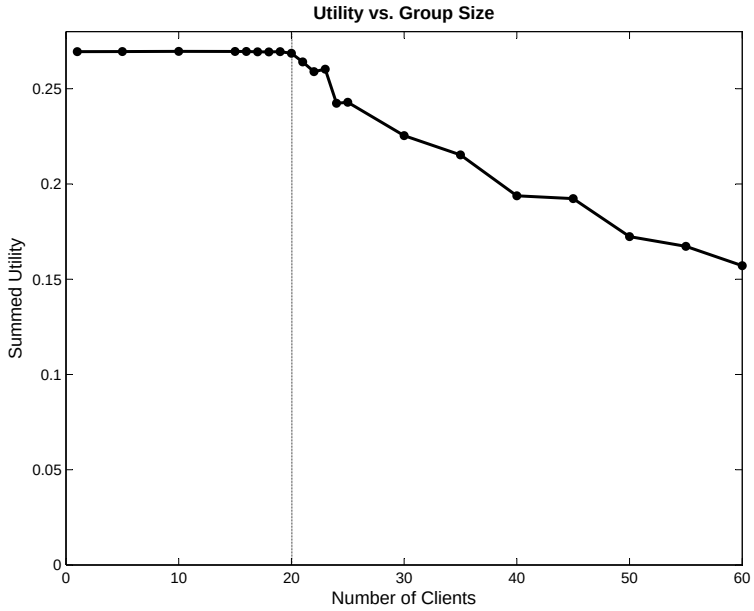

Fig. 6. This figure shows the average summed utility metric for several sessions. The sessions were performed with a variety of user group sizes ranging from one to sixty. Given the server bandwidth allocation of 100Mbps and a per-client bottleneck speed of 5Mbps, the drop off in performance at 20 clients is expected. This is because the system's overall performance is bandwidth limited.

twenty. The drop-off point is where we would expect given the server's link speed of 100Mbps and a per client link speed of 5Mbps.

In our future work, we will be addressing methods for even more scalable data distribution. One way to achieve this is to utilize multicast protocols which group users with similar data interests. A key question then becomes which data should be grouped (i.e. clustered) together into the same multicast channel. This motivates the experiments of the next subsection in which we look at how clustering affects adaptation performance.

5.3.4 *Impact of Clustering on Performance.* An important engineering parameter when mapping a specific dataset to our abstract representation graph is to specify the clustering of edges into units of data that are accessed atomically. For a given dataset, the range of options is quite wide.

At one extreme, every edge can be place within its own cluster. This allows each edge to be individually addressed, providing the maximal amount of data access flexibility. However, there is an increase in the required overhead in managing all of the cluster subscriptions.

At the other extreme, all edges are grouped into a single cluster. This is equivalent to simply downloading a single large file with a static data ordering. This does not provide any data access flexibility, but it dramatically reduces the amount of management overhead because there are no choices to be made.

In practice, it will be best to choose a middle ground between these two ex-

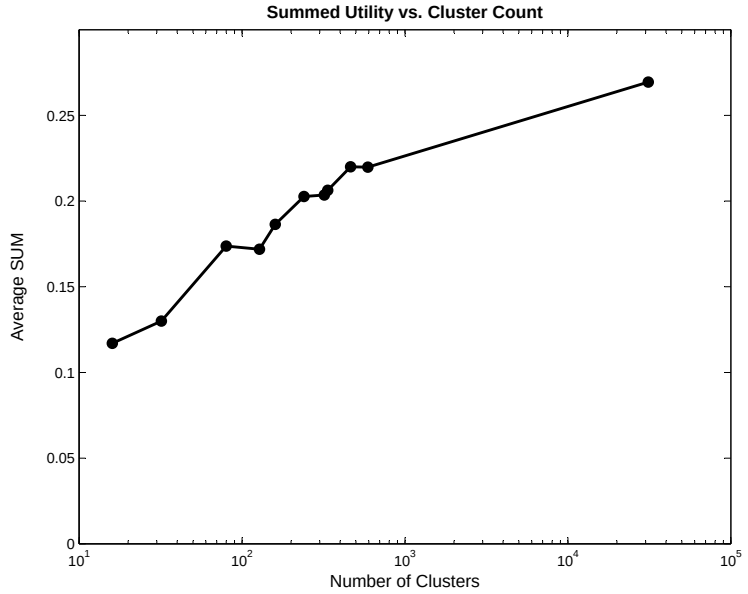

Fig. 7. This figure shows the impact of clusters on overall performance. As the number clusters decreases, so does the flexibility of data access patterns, resulting in a drop of the average SUM value. We call this the clustering penalty. However, the clustering penalty is not evenly distributed. The penalty is greatest when there are a very small number of clusters. As the number of clusters grows, the penalty gets smaller and smaller. This leads to the linear shape of the plot using a logarithmic scale for the number of clusters.

tremes that provides adequate flexibility without incurring too much overhead. We designed an experiment to explore this middle ground by varying the number of clusters present in the representation graph. The results are shown in Figure 7.

The results show that as the number of clusters drops, the average SUM value decreases. For all sessions, we simulated a single client with a fixed bottleneck bandwidth of 5Mbps. The only setting that changed between sessions is the number of clusters, and the drop in utility is a direct result of this clustering parameter. We refer to the drop in utility as the *clustering penalty*.

It is important to note that the graph shows the number of clusters using a logarithmic scale. Therefore, the linear shape of the plot indicates a substantial clustering penalty only for configurations with relatively few clusters. Additional clusters will improve quality by increasing the flexibility of data access. However, there is a decrease in the marginal benefit of increasing the number of clusters as the cluster count grows. This result implies that a small amount of clustering to reduce communication overheads can be used without seeing a dramatic impact on system utility.

6. CONCLUSIONS AND FUTURE WORK

We have presented our layered system design of GAL: A Generic Adaptation Library. GAL is a middleware library built to realize the conceptual adaptive framework outlined in our previous work [GMP04]. Our design specifies three primary

system layers: the application layer, the adaptation layer, and the communication layer.

The GAL library falls within the adaptation layer, between the low level network code of the communication layer and the high level code of the application layer. We defined a series of application programming interfaces at the borders between layers which delegate responsibility for various functions to the appropriate layers.

We introduced the Summed Utility Metric (SUM) as an application-independent measure of system performance. The SUM is based only on the status of the representation graph and the same utility metric used to drive the adaptation process.

Finally, we performed a series of evaluations that show that GAL can effectively adapt to changing system conditions. Using the SUM metric, we explored the impact on performance of several factors including bandwidth availability, group size, and edge clustering.

Given the success of our experimental prototype, there are several promising directions for future work. First, we would like to explore methods for enabling large user groups to access similar datasets from a single server. We feel that a scalable solution can be developed by clustering data into larger atomic units and designing an efficient communication layer. This type of solution would take advantage of the relatively small cluster penalty when the number of clusters is large.

A second promising area for future exploration is in designing a "tool box" of cost and utility metrics to include with our standard GAL library. We believe that a small set of metrics would be powerful enough to express the adaptation needs for number of adaptive applications. Providing a standard tool box of these metrics could speed development times for these complex systems.

Finally, we would like to streamline our GAL implementation and make it available as an open-source project for the community to use and integrate into their ongoing work. In our own lab, we are already incorporating GAL into a number of projects. In the future, we hope to be ready to share our tools with others in the hopes of developing even better algorithms and methods for multidimensional adaptation.

REFERENCES

Daniel G. Aliaga, Thomas Funkhouser, Dimah Yanovsky, and Ingrid Carlbom. Sea of images. In *Proc. of IEEE Visualization*, 2002.

Shawn Bowers, Lois Delcambre, David Maier, Crispin Cowan, Perry Wagle, Dylan McNamee, Ane-Francoise Le Meur, and Heather Hinton. Applying adaptation spaces to support quality of service and survivability. In *DARPA Information Survivability Conference and Exposition*, 2000.

Saurav Chatterjee, Jerry Sydir, Bikash Sabata, and Thomas Lawrence. Modeling applications for adaptive qos-based resource management. In *Proc. of IEEE High Assurance Systems Engineering Workshop*, 1997.

L. D. Floriani and P. Magillo. Regular and Irregular Multi-Resolution Terrain Models: A Comparison. In *Proc. of 10th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'02)*, pages 143–148, 2002.

David Gotz and Ketan Mayer-Patel. A General Framework for Multidimensional Adaptation. In *Proc. of ACM Multimedia*, New York, NY, USA, 2004. Association for Computing Machinery.

David Gotz and Ketan Mayer-Patel. A framework for scalable delivery of digitized spaces. *In-*

ternational Journal on Digital Libraries, 5(3):205–218, May 2005. Special Issue on Digital Museums.

David Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, pages 24–35, May 2001.

R. Rejaie, M. Handley, and D. Estrin. Quality adaptation for congestion controlled video playback over the internet. In *Proceedings of ACM SIGCOMM*, pages 189–200, 1999.

Jonathan Walpole, Charles Krasic, Ling Liu, David Maier, Calton Pu, Dylan McNamee, and David Steere. Quality of service semantics for multimedia database systems. *Database Semantics: Semantic Issues in Multimedia Systems*, 1999.

Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.