Resource Management for Responsive Web Computing

Azer Bestavros, Marina Chen, Mark Crovella,
Abdelsalam Heddaya, and Stan Sclaroff
Boston University Computer Science Department

James Cowie
Cooperating Systems Corporation

March 21, 1996

1 Introduction

The exploding demand for services like the World Wide Web reflects the potential that is presented by globally distributed information systems. The number of WWW servers world-wide has doubled every 3 to 5 months since 1993, outstripping even the growth of the Internet. At each of these self-managed sites, the Common Gateway Interface (CGI) and Hypertext Transfer Protocol (HTTP) already constitute a rudimentary basis for contributing local resources to remote collaborations.

However, the Web has serious deficiencies that make it unsuited for use as a true medium for *metacomputing* — the process of bringing hardware, software, and expertise from many geographically dispersed sources to bear on large scale problems. These deficiencies are, paradoxically, the direct result of the very simple design principles that enabled its exponential growth.

There are many symptoms of the problems exhibited by the Web: disk and network resources are consumed extravagantly; information search and discovery are difficult; protocols are aimed at data movement rather than task migration, and ignore the potential for distributing computation. However, all of these can be seen as aspects of a single problem: as a distributed system for metacomputing, the Web offers **unpredictable performance and unreliable results.**

Responsive Web Computing The goal of our project is to use the Web as a medium (within either the global Internet or an enterprise intranet) for metacomputing in a reliable way with performance guarantees. We attack this problem one four levels: (1) network services and protocol-level techniques, (2) middleware solutions such as caching, prefetching, and replication, (3) Web computing resource management models, protocols, and services, and (4) an object-oriented framework to capture these models and associated protocols and services along with application-specific knowledge and the overall designs of Web computing applications.

Resource Management Services Globally distributed computing allows novel approaches to the old problems of performance guarantees and reliability. Our first set of ideas involve setting up a family of real-time resource management models organized by the Web Computing Framework with a standard Resource Management Interface (RMI), a Resource Registry, a Task Registry, and resource management protocols to allow resource needs and availability information be collected and disseminated so that a family of algorithms with varying computational precision and accuracy of representations can be chosen to meet realtime and reliability constraints.

Middleware Services Complementary to techniques for allocating and scheduling available resources to serve application needs under realtime and reliability constraints, the second set of ideas aim at reduce communication latency, traffic conjestion, server work load, etc. We develop customizable middleware services to exploit application characteristics in traffic analysis to drive new server/browser design strategies (e.g., exploit self-similarity of Web traffic), derive document access patterns via multiserver cooperation, and use them in speculative prefetching, document caching, and aggressive replication to reduce server load and bandwidth requirements.

Communication Infrastructure Finally, to achieve any guarantee of quality of service or performance, one must get at the network layer that can provide the basic guarantees of bandwidth, latency, and reliability. Therefore, the third area is a set of new techniques in network service and protocol designs.

Object-Oriented Web Computing Framework A useful resource management system must deal with job priority, fault-tolerance, quality of service, complex resources such as ATM channels, probabilistic models, etc., and models must be tailored to represent the best tradeoff for a particular setting. This requires a family of models, organized within an object-oriented framework, because no one-size-fits-all approach is appropriate. This presents a software engineering challenge requiring integration of solutions at all levels: algorithms, models, protocols, and profiling and monitoring tools. The framework captures the abstract class interfaces of the collection of cooperating components, but allows the concretization of each component to be driven by the requirements of a specific approach and environment.

Reuse, Verification, Technology Transfer It is important to emphasize that though we frame the Web computing resource management problems in the above three complementary areas, the solutions and results for each area stand on their own and are useful independent of the other areas.

To demonstrate our ideas and the capability of Responsive Web Computing System, we will take an existing Web image search engine that was implemented on a centralized server and come up with a new distributed design to serve as our Web computing focus application. Such a distributed image search engine for globally-distributed image databases has direct application in areas of national strategic importance such as defense tactical image analysis, national security, intelligence gathering, and crime prevention.

Last but not least, to make both the image search engine and the Responsive Web Computing System available to the research community, we will publish the frameworks and concretization of their components over the WWW. We will be using incremental framework development tools and graphical user interface tools from Cooperating Systems Corporation to visualize and monitor all the distributed players that fill the roles of the abstract frameworks, and provide point-and-click version control over the distributed software base. The RWC project will deliver the first usable Web-based, Web-available software testbed for real-time, reliable, wide-area collaborative computing.

In the following sections, we describe the major components of RWC, followed by related work.

2 Responsive Metacomputing

The goal of Metacomputing is to add the necessary functionality to large-scale distributed systems—such as the Internet[1]—to support the demands of High Performance Computing (HPC) applications.

Therefore, the goal of Metacomputing is *not* to reinvent HPC for the Internet. Such an approach has the obvious drawback of confounding the already difficulty task of parallel programming over a dedicated network with a whole new set of complexities: heterogeneity, unreliability, long latency, and total lack of control of underlying resources. Instead, the goal of Metacomputing is to adapt the Internet—by improving its performance and predictability—so that it could be used to support the solution of large scale problems which in turn may require HPC application components.

An example of a metacomputing application is FAFNER [24], an effort that brings resources and expertise from many sites world-wide to solve the problem of factoring RSA-130. The approach that FAFNER uses to harness the Internet resources available at its disposal to factor RSA-130 can be best described as a *best-effort* approach, which does not offer any guaranteed performance. While useful for a variety of applications, *best effort* Metacomputing is not sufficient for applications that are subject to timing and reliability constraints. Examples of such applications include: interactive applications (*e.g.*, battlefield group simulations), time-constrained database queries for real-time applications (*e.g.* image search for tactical image analysis), and applications that involve temporal data (*e.g.* stock market modeling and weather prediction). The real-time and reliability constraints imposed by these (and other) applications require *responsive* rather than merely *best-effort* Metacomputing.

[1] In the remainder of this section, we will use the term "Internet" to mean either *the* Internet or the intranet of a particular enterprise

The *best-effort* philosophy of current Metacomputing platforms is due to the unpredictability of the underlying computing infrastructure, which is due to two reasons: (1) the inability of applications to control or negotiate the resources they need, and (2) the lack of predicability at the network transport level. Therefore, in order to achieve responsive Metacomputing, new protocols and services need to be developed to reduce this unpredictabilty and to allow a certain level of *commitment* when resources are contributed to a Metacomputing platform.

Another problem which could limit the wide-spread use of Metacomputing is the sheer amount of software development time involved in solving any large scale problem. Therefore Web computing must leverage existing HPC efforts by (1) reusing the HPC software base as components when appropriate (*e.g.*, running a CFD component in a large manufacturing Web computing application), and (2) providing a software engineering paradigm for the design and integration of software components (of which as many as appropriate are from an existing code base) that will reduce the development cycle through the promotion of standard interfaces and common programming methodologies.

To tackle the above issues, our research will focus on building a *Responsive Web Computer* (RWC) by developing the necessary technology in the four levels of abstraction shown in Figure 1. In particular, our work will focus on:

1. Developing object-oriented frameworks for Webcomputing resource management and Webcomputing applications. The Web computing Resource Management Framework will provide standard interfaces between application programs and the RWC resource manager to capture the design in which the RWC services, middleware, and protocols are put together for a variety of objectives such as timeliness and fault-tolerance. In addition, a Webcomputing application framework will be developed for our focus application WebSearch, to demonstrate how application-specific knowledge can affect the effectiveness of resource management.

2. Developing a pallette of Metacomputing Resource Management Services to improve the predictability of the distributed information system underlying the RWC through the management (registration, admission control, and scheduling) of available system resources to ensure responsiveness (timeliness and fault-tolerance).

3. Developing Middleware Services—common services such as caching, replication, prefetching, performance prediction and synchronization services—that could be customized to exploit application characteristics to provide scalability and improve performance.

4. Developing new transport protocols that alleviate the principal end-to-end performance and resource limitations of the RWC network infrastructure—whether that infrastructure is the Internet or the intranet of a particular enterprise. These techniques will be based on a careful analysis of traffic characteristics for RWC applications.

Research in each one of the layers shown in Figure 1 and described above is important in its own right, and should advance the state-of-the-art in Metacomputing technology. However, there is much to be learned from the interaction amongst these layers to provide holistic solutions to the many difficult metacomputing problems. To understand and exploit such interactions, we intend to incorporate the algorithms, protocols, and services developed in all four levels enumerated above into a single prototype RWC, which will serve as a Metacomputing testbed for HPC applications in general—and for an Internet image search application for tactical picture analysis in particular.

In the remainder of this section, we examine the four basic layers of our RWC shown in Figure 1. We start with the bottom three layers (resource management, middleware services, and transport protocols), which address the issues of predictability, scalability, and end-to-end performance, respectively. Next, we examine our RWC object-oriented software development paradigm, which abstracts away details of the RWC internals and promotes software reuse. We conclude this section with an overview of an example RWC application: An Internet image search application for tactical picture analysis.

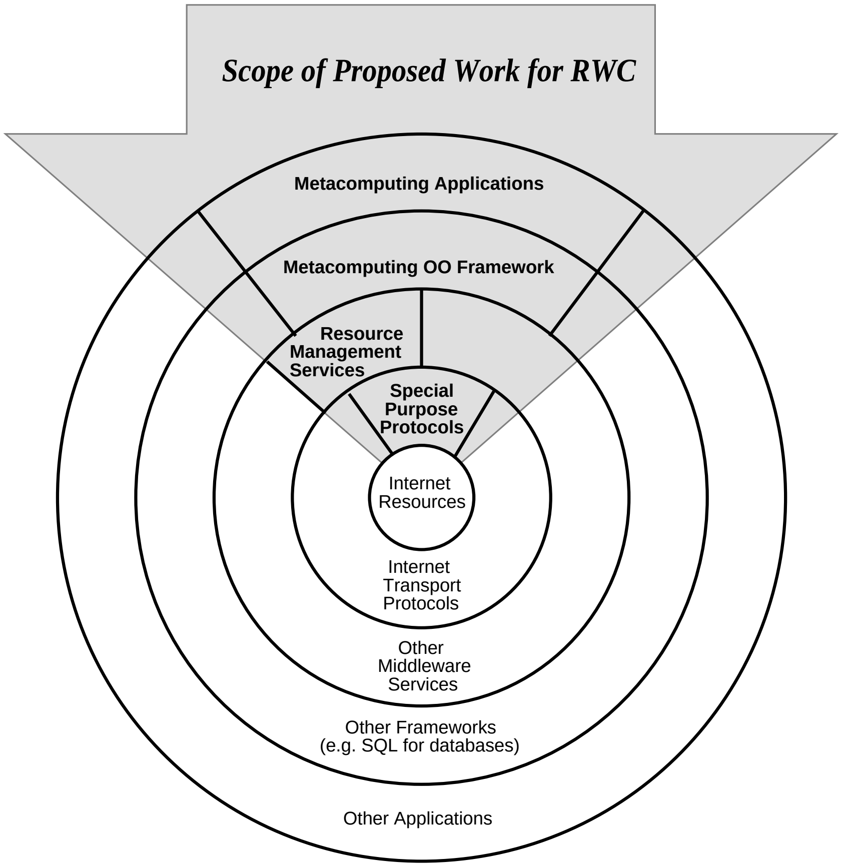
Scope of Proposed Work for RWC
Metacomputing Applications
Metacomputing OO Framework
Resource Management Services
Special Purpose Protocols
Internet Resources
Internet Transport Protocols
Other Middleware Services
Other Frameworks (e.g. SQL for databases)
Other Applications

Figure 1: Layers of a Responsive Metacomputing Platform

3 RWC Resource Management Services

To support a RWC, resource management services must be developed to allow the assignment of computations to RWC resources in a way that satisfies the resource requirements of these computations for a predictable, responsive performance.

Resource Management Interface: The Resource Management Interface (RMI) is an abstraction that allows the computational requirements of RWC processes to be matched with the resources available at the disposal of the RWC through a schedule that satisfies the timing and fault-tolerance requirements of these processes.

The overall structure of the Resource Management Interface is shown in Figure 2. There are three main services to be supported. The Task Registration Service allows the computational resources needed by, and the performance constraints imposed on a RWC task to be specified. The Resource Registration Service allows the computational resources contributed to the RWC to be specified. The Resource Management Service provides the admission control and scheduling protocols for managing the registerd RWC resources in accordance with the performance constraints of the registered RWC tasks.

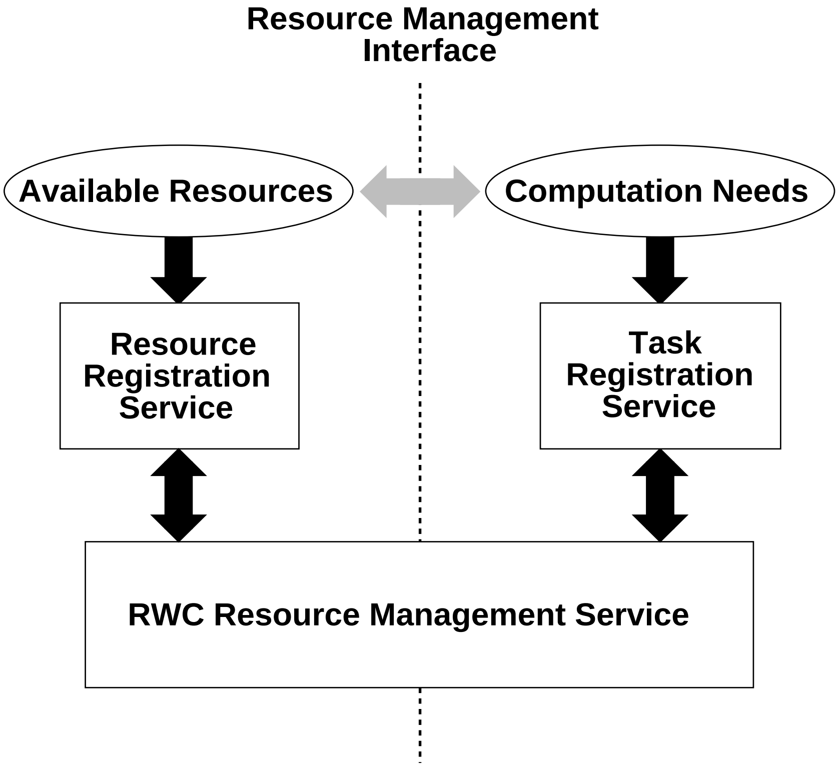
Resource Management Interface
Available Resources
Computation Needs
Resource Registration Service
Task Registration Service
RWC Resource Management Service

Figure 2: Components of the Resource Management Interface

3.1 Task Specification

To motivate the issues involved in the specification of a task to the resource manager, consider a simple situation where a real-time application is to be executed on the RWC. Furthermore, assume that the real-time application consists of a large set of periodic processes, where each process requires various resources (*e.g.* CPU cycles, network bandwidth, *etc.*) As a concrete example, consider a Web agent, which is responsible for monitoring the contents of a particular object (*e.g.* areal radar map, wheather map, number of objects in a given database, stock quote). Assume that the performance of that agent is constrained so as to report back the results of its operation periodically (say every minute) to another agent (*i.e.* another RWC task).

Furthermore, assume that it is requested that the agent's periodic behavior must be started within one hour time.

A possible representation of this application would be a set of tuples of the form $(P_i, R_i, S_i, C_i, T_i, L_i)$, where P_i is a process ID, R_i is a resource ID, T_i is the period of P_i, S_i is the startup demand of P_i from R_i, C_i is the cumulative demand of P_i from R_i per period T_i, and L_i is the latest time for starting the periodic component of P_i (in other words, it is the deadline imposed on finishing up the startup, or non-periodic component of P_i.)

To clarify this particular model, we look at some examples (see Figure 3 for an illustration). If R_i is a CPU resource, then S_i would be the CPU time needed to startup the periodic behavior of P_i, and C_i would be the CPU time needed by P_i every T_i units of time thereafter. The periodic behavior of P_i must start not later than time L_i. If R_i is a TCP/IP connection or an I/O device, then S_i would be the number of bytes to be transfered in order to startup P_i and C_i would be the number of bytes to be transfered periodically (every T_i units of time) thereafter. If R_i is a memory resource (*e.g.*, disk or main memory), then S_i would be the number of bytes to be reserved in order to startup P_i and C_i would be the extra number of bytes to be stored every T_i units of time. Having an application defined as a set of such tuples, one could talk about the requirements (or demand) of a particular component (process) or subset of components from the application.

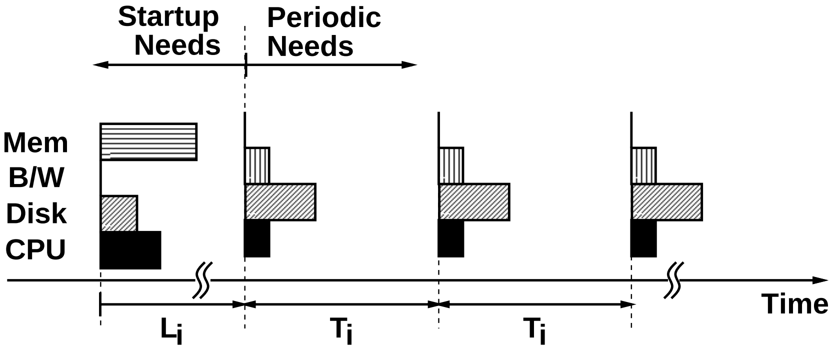

Figure 3: Specification of the Resource Requirements and Timing Constraints of a Task

The above model allows the specification of a real-time task as an aperiodic time-constrained *startup* subtask followed by a periodic subtask. This allows us to use the same model for the specificaton of both periodic and aperiodic tasks. Despite its simplicity, the above model allows for the specification of a large number of task requirements. For example, a tuple $(P_i, R_i, S_i, 0, \infty, \infty)$ specifies that process P_i needs S_i units from resource R_i, and that this need could be satisfied whenever possible since no timing constraints were attached.

3.2 Resource Specification

To motivate the issues involved in the specification of a resource to the resource manager, consider a simple situation in which a Web server is to be contributed to the RWC (say overnight), for a period of 12 hours. However, assume that this server is also expected to provide other services (*e.g.* FTP, HTTP, *etc.*). Rather than totally committing this server to the RWC (and thus making it unavailable for other tasks), it may be prudent to commit only (say) 50% of its CPU time to the RWC and only (say) 100 MB of its disk space, for example. This level of commitment may be different depending on the mode of the Web server's operation, *etc.*

A possible representation of resources contributed to the RWC would be to use tuples of the form $(Q_i, R_i, C_i, T_i, L_i)$, where Q_i is a Web Computer ID and R_i, C_i, and T_i are as before, except that instead of *demand*, C_i represents *supply* (or availability). L_i demarks the end of the commitment period.

One way of thinking about this model of RWC resources is to think of the contributor of the resource as specifying a *budget* C_i for the RWC to expend out of this resource until time L_i. This budget is replenished to C_i every P_i units of time. This is similar to the Deferable Server [87] and Sporadic Server [82] real-time scheduling techniques for aperiodic tasks, which are extentions of RMS theory [57]. The rational behind representing the availability of resources in this fashion is to allow RWC contributors to control the level at which their resources are to be committed to the RWC. In particular, by adjusting the ratio of C_i to T_i, a contributor controls the percentage of the resource capacity to be committed to the RWC. The intrusion of such commitment can be further tuned by adjusting the period T_i—in effect limiting the length of time the contributor's local processing would have to wait for RWC processes.

The above model allows for the specification of a large number of *commitment levels*. For example, a tuple $(Q_i, \texttt{CPU}, 0.5, 1, \texttt{TimeStamp})$ specifies that until $\texttt{TimeStamp}$, Q_i is contributing up to 0.5 seconds of its CPU every 1 second. A tuple $(Q_i, \texttt{CPU}, \infty, \infty, \texttt{TimeStamp})$ specifies that until $\texttt{TimeStamp}$, Q_i is contributing its CPU completely to the RWC.

The periodic underpinning of the above computation and resource models allows us to use well established algorithms and results from real-time scheduling research while not restricting the class of computations or resources to be handled to be periodic.

3.3 Resource Management Algorithms and Protocols

Once an appropriate interface is defined, the assignment of processes could be accomplished. For example, the process of finding a Web Computer that could satisfy the requirements of a particular process P_i amounts to searching for a contributor Q_j such that for every tuple $(P_i, R_i, S_i, C_i, T_i, L_i)$, there exists a tuple (Q_j, R_i, C_j, T_j) such that the following is true:

$$\begin{aligned} \lceil \frac{S_i}{C_j} \rceil \times T_j &\leq L_i - t && (1) \\ \frac{C_i}{T_i} &\leq \frac{C_j}{T_j} && (2) \\ T_j &\leq T_i && (3) \end{aligned}$$

In order to provide the resource management algorithm with the information it needs about processes and resources, it is necessary to devise a protocol for the collection/dissemination of such information. For example, one could use a hierarchical registration protocol such as the one implemented in FAFNER [24]. Alternatively, one could devise a more distributed *peer-to-peer* protocol, whereby resource availability is dessiminated passively (via gossiping [11]) or actively (via bidding [84]). The hierarchical approach has the advantage of being simpler to implement than the peer-to-peer approach. However, the peer-to-peer approach offers more resiliency against failures, *etc.*

3.4 Open Research Problems

The simplistic resource management model and algorithm introduced above is just to motivate and illustrate the issues involved in resource management for a RWC. A realistic model (such as the one we propose to develop and implement as part of this proposal) will have to consider many more issues. Examples of such issues include:

Dealing with priority issues: Tasks in a mission-oriented environment are typically prioritized (or time-constrained) to reflect a particular mode of operation. When the system's mode changes, dynamic adjustment of task priorities (and/or time constraints) may be necessary.

Dealing with different deadline semantics: There are several deadline semantics for real-time tasks (*e.g.*, soft, firm and hard deadlines). Ultimately, each task is assigned a *value function*, and the goal of the system becomes that of maximizing profit (or minimizing loss).

Dealing with inter-process synchronization issues: Typically, the execution of a group of tasks must be synchronized to achieve a desired behavior (*e.g.*, Communication, Producer/consumer issues). Such synchronization constraints must be possible to specify to the resource management layer, and must be reflected in the eventual scheduling of tasks to resources.

Dealing with fault-tolerance requirements: Tasks in a mission-oriented environment may differ in the level of fault-coverage they may require (*e.g.* failure masking, failure detection, and failure recovery). Such fault-tolerance requirements must be possible to specify to the resource management layer, and must be reflected in the eventual scheduling of tasks to resources (*e.g.*, using NMR techniques to achieve failure masking).

Dealing with resources from multiple platforms: In order to manage resources from different platforms, there is an obvious need to develop metrics that could be normalized through the use of simple benchmarks when a resource is registered (*e.g.* metrics to gauge CPU time for different architectures).

Defining/classifying more complex resources and processes: The discussion above assumed simple resources such as CPU and Memory and simple processes such as periodic or aperiodic processes. More complicated models may be needed for more complex resources and processes (*e.g.* tasks that allow imprecise results).

4 RWC Middleware Services

The explosive growth in availability and use of the Internet demands that we raise the level of common services and introduce new types of higher-level services. These common services would lie between the transport and application levels, hence the term *"middleware"* [68], and would provide means for extending commonly available services on the network to include new abstractions.

Our research in Middleware services will focus on two critical aspects of the RWC: 1) Improved scalability of information retrieval for RWC applications through proper partitioning and distribution of data using caching, replication, and prefetching techniques; and 2) Improved predictability of the RWC through efficent performance prediction services and load-balancing protocols. The remainder of this section explores these middleware services, presenting our previous work in each area and our proposed research building on that work.

4.1 Services for Scalable Information Retrieval

Previous work by the Oceans group has singled out caching and prefetching as important components of the middleware infrastructure. Current caching and prefetching protocols are client-based; they do not make use of the knowledge amassed at servers about client access patterns. In our WebSeed project, we have demonstrated that these myopic solutions, focussing exclusively on a particular client or set of clients, do not scale. We have shown this through a pilot study of WWW client-based caching [9], in which client traces [28] were collected and used to drive simulations of various client-based caching protocols. Our work in [8] demonstrated that the server knowledge of temporal, spatial, and geographical locality of reference could be used effectively to improve end-to-end performance. In that respect, we identified two server-initiated mechanisms that make use of such knowledge, namely data dissemination [5] and speculative service (or server-initiated prefetching) [6]. Using extensive trace simulations we quantified potential gains of up to 40% in network traffic, 42% in server load, and 50% in service time.

There are two components of our proposed work in middleware services for scalable information retrieval. First, we propose to further our research in information dissemination and speculative service protocols. One important extension of our work will be to allow for middleware services to be initiated by multiple *cooperative servers*—as opposed to our current approach which confines the dissemination and speculative service to a *single* server. For example, our speculative service research has concentrated so far on exploiting the spatial locality of reference that may be present between documents on a single server. By allowing servers to exchange statistics about client access patterns, it is possible to exploit the spatial locality of reference that may be present between documents on multiple servers. Similar inter-server extensions are possible for dissemination and caching middleware services. Another important extension of our work would be to study and exploit the interaction between various middleware services. For example, our dissemination protocols do not assume the existence of any considerable client/proxy caching. The effect of such protocols is to distort the temporal, geographical, and spatial locality of reference conclusions at the server, which may negatively affect replication and dissemination decisions. Another example of inter-service effects involve the benefits that are possible when server-initiated and client-initiated prefetching protocols are coordinated as proposed in [6, 10], or when server-initiated prefetching and client caching protocols are coordinated as proposed in [6]. Second, we propose: 1) the development of prototype dissemination and prefetching middleware protocols, and 2) the integration of these protocols within Internet applications. In particular, we propose to extend the HTML language to allow middleware services at a server to convey *"hints"*—meta information about prefetching and dissemination—to clients. This would require the development of middleware service components at both the server and client ends. For example, at the server end middleware utilities would generate and maintain the databases necessary for dissemination and for communicating prefetching hints. At the client end, middleware utilities would aggregate statistics to assist servers in (say) replica placement decisions.

4.2 Services for Performance Prediction

Our group has shown that significant performance improvements can be secured if Internet applications make use of sophisticated middleware services that involve dynamic performance measurements. For example, we have shown that one of the more effective ways to decrease the latency of information transfer in the Internet is for applications to *actively avoid congested paths.* This observation is based on careful study of network traffic and understanding of the time-varying nature of congestion.

To demonstrate application-level congestion avoidance, we have proposed and evaluated an application behavior called *dynamic server selection*[27], which stands in contrast to typical systems, in which the existence of a single *"best"* place for service (*e.g.*, information retrieval) is assumed. In our system, there is no best server; instead the application is empowered—through the use of appropriate performance prediction middleware services—to select the appropriate server at the latest possible moment. Dynamic server selection relies on minimal-impact network software probes to assess the congestion present over links to various servers. Currently we use the *ping* tool, which employs ICMP ECHO packets, to measure round trip time as our metric of link condition (congestion). While we have shown that a ping packet yields excellent results for predicting the transfer time of small files (whose transfer time is dominated by round trip time), we have also found that as file size grows, it becomes important to include an estimate of the maximum possible bandwidth available, and of the current bandwidth avaiable between client and server, to more accurately estimate the document transfer time.

We propose to implement two powerful tools for link condition measurement, which we call *Bprobe* and *Cprobe.* Bprobe will use a *sequence* of ECHO packets to measure the raw or link bandwidth available between client and server (using the principles outlined in [13]). Cprobe will use the information provided by Bprobe, along with a simple stream of probe packets, to discover the amount of intervening traffic between client and server. Together these tools will provide an estimate, not just of round-trip time, but also of the maximum available bandwidth and of the current available bandwidth between client and potential server. Using Bprobe and Cprobe it will be possible to account for the bandwidth limitations that occur in transferring large files through the Internet, and hence will improve our scheme for dynamic server selection. Also, Bprobe and Cprobe will have utility in their own right, wherever knowledge of network conditions is needed on a dynamic basis.

4.3 Large scale parallel distributed computing

The challenge of parallel programming compounds when coupled with the economic necessity of harnessing autonomous workstations, that are connected via relatively slow general purpose networks. Our proposed work concentrates on the second problem, that of relatively slow communication and synchronization. We aim to provide a small, but open, set of core parallel programming primitives, in the form of an extension of the MPI standard interface. These primitives support both message passing and shared memory in a manner that enables the programmer to treat the code that lies between synchronization points as if it were purely sequential. At the same time, our proposed system will be able to guarantee a minimum parallel efficiency, at the cost of a controlled increase in granularity of parallelism.

Over the past several years, we have worked on Mermera [42, 41, 40, 43], a parallel programming library that provides coherent shared memory in software for programming convenience, as well as seamlessly integrated *noncoherent* shared memory for performance. Noncoherence plays the same role in shared memory systems that asynchrony and unreliability play in message passing systems. By commingling the two classes of memories in a disciplined manner, Mermera enables parallel program development by step-wise refinement, or selective optimization. The programmer can prototype a parallel program using coherent shared memory, then convert critical subprograms to employ noncoherent shared memory, choosing the subprograms that can safely exploit the resultant speed and fault-tolerance. Our work in Mermera, as well as other people's work on related systems [56, 1, 96], provides specific guidance in the exploitation of noncoherent shared memory.

Our current project, called BSPk (for Bulk-Synchronous Parallel toolkit), builds on the success of Memera in achieving its goals. Mermera provides effective system support to *mask memory distribution* from the programmer, at the cost of introducing noncoherence. BSPk complements this success by constructing system facilities for *masking concurrency* efficiently. In 1990, Valiant [95] proposed the BSP model for parallel algorithms, with such attractive features as feasibility of implementation on realistic hardware, and tractability of complexity analysis.

A program in BSP consists of a sequence of *supersteps* separated by barrier synchronizations (which can be expensive—more on that below). Each superstep contains a minimum number of computation steps, and a maximum number of communication steps, be they messaging or remote memory operations. The model offers concrete guidelines for system software to compute and enforce these bounds, and obtain a consequent guarantee on the efficiency of the entire computation. A central component, therefore of BSPk, is the support of *multithreading.* By automatically allocating and migrating threads, BSPK can precisely ensure the conditions for high efficiency stipulated by the BSP model. From the point of view of programmers, BSP offers the immense simplifying opportunity for the programming model to tie the semantics of communication to that of barrier synchronization in such a way as to completely mask concurrency during each superstep. We achieve this in BSPk—which supports both message passing and distributed shared memory—by ensuring that communication steps appear to occur exactly on the dividing line between supersteps. This idea follows in the line of such notions as critical sections and atomic transactions.

Everything else in BSPk is designed for extremely low communication and synchronization overhead. For example, we insist on BSPk handling memory allocation of message buffers, so as to carry out *application level framing* of messages [23], and consequently achieve zero-copy transmission. Similarly, we include directives for the programmer to predeclare the expected communication pattern when known, thus enabling *zero-overhead barrier* synchronization by message counting.

5 RWC Communication Infrastructure

The Resource Management Services described above are sufficient to provide performance assurances for the RWC. However, the *quality* of those performance assurances is critically dependent on the reliability and performance of the underlying infrastructure. That is, while the Resource Management interface and algorithms will provide RWC applications with a responsive computing environment, the performance guarantees in such an environment may be greatly improved by modifications to the underlying data transport mechanisms. Thus, while not strictly necessary for implementation of the RWC, we feel that improved communication mechanisms will vastly increase the utility of the resulting RWC.

To improve the performance guarantees of the RWC the issue of "end-to-end" network performance becomes critical. The objective metric in end-to-end analysis is the *application-perceived* performance of the RWC—the responsiveness of the system in terms of latency of information access and rate of information delivery.

Previous attempts to improve the end-to-end performance of the Internet have primarily focused on flow-control mechanisms. Such mechanisms emphasize an idealized, "steady-state" view of network traffic [46, 15]. In contrast, our recent work (as well as that of others) has shown that network traffic is fundamentally more variable that was previously assumed. This variability means that flow-control protocols whose goal is the acheivement of an ideal, steady-state condition are bound to be unreliable in the real Internet.

As a result, properly addressing the end-to-end performance of the RWC will require improvements at two levels — modeling of network traffic characteristics, and transport protocol design. At the lower level, studying the nature of network traffic is essential to understand how available network bandwidth and latency varies in the Internet; and at the transport protocol level, TCP/IP flow control mechanisms will need to be redesigned in a number of ways to accomodate the special nature both of competing traffic on the Internet and of RWC traffic. The remainder of this section explores these two levels, presenting our previous work in each area and how our new understanding of the dynamics of real networks requires a fundamental re-design of transport protocols for the RWC.

5.1 Network Traffic

In previous work we have found that network traffic generated by the World Wide Web is *self-similar.* Self-similarity—a property associated with fractals (objects whose appearance is unchanged regardless of the viewing scale)—has been rigorously established for Ethernet traffic [54]. Since a self-similar process has observable bursts on all time scales, it exhibits *long-range dependence.* The significance of this property is beginning to be observed in studies such as [53, 64], which show a radical difference between packet loss and delay behavior in simulations using real traffic data and those using traditional (Poisson or Markov) traffic models. Moreover, the presence of long-range dependence means that measurements of network traffic have a slowly-decaying autocorrelation function. As a result, it is possible to use past traffic measurements to obtain much better predictions of future traffic patterns than would be possible under traditional prediction models.

The fundamental contribution of our work to date is two-fold: 1) We have shown that WWW traffic shows strong indications of self-similarity; and 2) we have shown that self-similarity in general may arise from a surprising direction: the distribution of file sizes being transferred over the network [25, 26, 65]. This latter conclusion draws a causal connection between the *heavy-tailed* distribution of WWW file sizes and the self-similarity of WWW network traffic. This is interesting because heavy-tailed distributions of "information chunks" are common in information science (*e.g.*, the distribution of book sizes and word sizes are typically heavy-tailed [59]).

This result has implications for the RWC. If the size distribution of objects being transferred in the RWC is heavy-tailed, then RWC traffic is expected to be self-similar. Initial results reported in [26] indicate that this may well be the case. If so, such a far-reaching conclusion lays the groundwork for a self-similarity-based approach to modelling the nature of RWC traffic. To generate such a model, we will undertake a rigorous study identifying the precise levels of self-similarity to be expected in RWC traffic. Such a study will examine the effects of network protocols, caching, and user preferences, along with the size distribution of RWC objects.

Based on the results of the previous step, we will develop a model of RWC network traffic. This model would be parameterized by system type, transport model, and network bandwidth and delay characteristics. This would serve as a basis for predicting the performance of information transfers needed at higher levels (transport and application). The resulting model for RWC traffic will be much more accurate than traditional network traffic models because of the additional predictive power provided by self-similar models, and because of the full understanding of RWC objects (code and data) that will be obtained in our study.

5.2 Transport Protocol Layer

A more significant result of our improved understanding of traffic self-similarity lies in its implications for the design of transport protocols with improved reliability characteristics.

The self-similarity of network traffic that has been observed in the wide-area Internet[66] means that at no observable level can traffic levels be considered to be "smooth." In particular, looking at the traffic flowing along any well-traveled link in the Internet, the traffic levels fluctuate significantly at all time scales.

The implication of this observation is that the application-perceived rate of data flow in a TCP stream is expected to be highly variable under typical conditions — since each individual TCP stream shares a fixed-bandwidth substrate with self-similar competing traffic. Thus, fundamental redesign of the method of transporting data in the Internet is necessary if we want to significantly improve the reliability of that data transfer.

Another important result of our previous work is the observation that, in the World Wide Web, most transfers are of extremely short duration[26, 28]. This is another dimension in which the TCP "steady-state" assumption breaks down: most connections exist for too short a time to achieve steady-state even if it were possible. The proportion of traffic in the Internet attributable to the WWW is increasing rapidly, and anecdotal evidence indicates that this “elephants and mice” distribution of transfer lengths is becoming the norm for the Internet as a whole.

These two observations (self-similar competition and elephants/mice distribution) lead to two methods for radically redesigning transport protocols to achieve reliability and high performance in the RWC.

First, we will develop an *Information Dispersal Transport Protocol* (IDTP) to achieve reliable throughput in the presence of self-similar traffic variation. Information Dispersal is the principle behind the use of controlled redundancy to improve fault-tolerance and timeliness of information systems (as used in error-correcting codes and RAID disk systems).

The Information Dispersal approach, while extremely novel, is quite feasible based on our previous experience with these techniques. The particular approach we take is called the Adaptive Information Dispersal Algorithm (AIDA) [4]. AIDA is a novel dynamic bandwidth allocation technique that makes use of minimal, controlled redundancy to guarantee timeliness and fault-tolerance up to *any* degree of confidence. AIDA itself is an elaboration on Rabin's IDA [69]. In [3], we tackled the end-to-end real-time flow control problem by using the AIDA, and previously we have shown AIDA to be a sound mechanism that improves considerably the performance of I/O and storage systems [2, 7].

For the RWC, we will develop a transport protocol (IDTP) that uses AIDA to encode data before transmission. The improved reliability of IDTP derives from the necessity to only receive a fixed fraction of packets in order to reconstruct the entire data object. These packets travel different paths, and so some will likely travel over fast paths, leading to significantly improved reliability and delay characteristics.

Our second transport redesign will address the steady-state problems when transporting small data objects (such as code and computations). Our approach is twofold: first, aggregate the TCP/IP control parameters (*e.g.*, window sizes) of connections sharing partial routes. Second, recognizing that steady-state often cannot be achieved for small objects, adopt a “greedy” approach that minimizes transmission time for most objects.

Aggregation of window control parameters is fruitful because of the locality properties of many sets of transmissions. In the RWC, although data objects may often be small, transmissions will likely take place repeatedly along set paths. The characteristics of those paths (round trip time, bandwidth) are normally estimated accurately only over a long TCP transmission. In the RWC however it will be feasible to develop estimates of path characteristics by spanning measurements from many connections, since connections will travel the same paths repeatedly. As a result better estimates (and therefore better throughput) can be obtained than is available when considering each connection in isolation.

In addition, flow control in the RWC should be optimized for the observed characteristics of the objects being transferred. In the case when most objects being transferred are small, it will be desirable to adopt a hybrid flow control approach: optimized throughput for large transfers, and optimize latency for small transfers. This implies that small transfers should effectively attempt to use maximum possible transmission rate. This is a radical approach to flow control but becomes more reasonable when one considers that dropping a single packet from a small transmission is roughly equivalent in performance to dropping the whole transmission, and that small transmissions by themselves are not likely to overload intermediate links (since, unlike the Internet of 5 years ago, intermediate links are often as fast or faster than the local network). Thus, maximum rate should be attempted for small transmissions, while a flow-controlled rate should be used for large transmissions.

The sum total of these changes to TCP, resulting in IDTP, represent a truly radical revision of current approaches to transport protocols in the Internet. However, these approaches are warranted and necessary based on our understanding of traffic in the real Internet, and our need for improved reliability of the network infrastructure in the RWC. While these transport protocol changes are not strictly necessary for the deployment of the RWC, their addition and use where possible will significantly improve the performance and utility of the resulting RWC.

6 Object-Oriented Metacomputing Framework

A framework is a set of cooperating abstract classes that make up a reusable design for a specific class of software. The classes are then instantiated by concrete code to form a particular application.

6.1 Frameworks versus Toolkits

For highest performance, supposedly generic services often need to take advantage of features of the application context in which they are invoked. This presents some problems for traditional library or toolkit-based approaches, because of the difficulty of designing a toolkit interface which is sufficiently general, yet capable of accepting detailed hints. One easy solution exposes the internals of client classes to the services they call, but this violates encapsulation and makes reuse incredibly difficult.

By contrast, the framework approach inverts this problem. The framework defines a set of abstract classes which cooperate to achieve some goal, or implement a family of similar applications. These classes offer each other the service—not generically, as in a simple toolkit, but according to the intercomponent protocols and relationships defined within the framework.

6.2 Why OO Framework

For traditional HPC on MPP or SMP, efficient use of computation resources is by and large a problem of finding good data layout to reduce interprocessor communication and using the right instructions in the right order for the most frequently invoked codes. Therefore, libraries make sense as a way for software reuse, with handcrafted, highly efficient subroutines that implement many important functions, (*e.g.*, numerical linear algebra, aggregate communication routines) used by many applications.

Metacomputing is built on top of HPC, geared towards large scale, heterogeneous, and possibly geographically dispersed applications. In other words, a metacomputing program is a collection of coarse-grained components, which themselves can be HPC programs running on MPP's or SMP's. Thus metacomputing resource managements need to focus on the interplay of components; optimized solutions for their interaction are highly dependent on the application structure. The use of frameworks is the right approach because it abstracts the outer level designs that put the pieces together. The actual component implementations can vary as long as they can use the same interfaces (defined by abstract classes). We see libraries and frameworks as complementary approaches needed by metacomputing, where the former aims at providing high quality HPC components and services cross-cutting many applications, while the latter aims at capturing the overall design of high level component interactions which take advantage of the characteristics and structure of a particular application class.

Framework plays a dual role in this project: First, it captures the complex, dynamic interactions between metacomputing services, the underlying resource management machinery, and the application running on the RWC. Second, it promotes the reuse of the overall design of a distributed WebSearch engine—our chosen demonstration application. In the following, we discuss these two frameworks.

Resource Management Framework (RMF): Figure 2 shows the components of the RMF at various layers. The top level consists of a Resource Management Interface, a Resource Registry, a Task Registry, and a Resource Manager. The Resource Management Interface (RMI) is responsible for communicating information about the computing tasks to the Resource Manager. This interface can be as narrow as a pair of integers as in the case of FAFNER's factoring tasks [24]. In general, such information can be conveyed as application attributes which constitute a so called *"little language"* for a class of applications. A concretized RMI will provide the necessary services to parse such little languages.

The Task Registry is responsible for registering all the the computation tasks needed to be performed by an application. In FAFNER, the entire factoring computation is represented as an interval, and subtasks are represented by their subintervals [24]. In general, there can be many models for resource managment accommondating different objectives, such as real-time constraints, fault-tolerance, scheduling priority, multi-precision tasks, etc. So the Task Registry may have different forms depending on the particular model(s) used, which will be the second layer components of the Framework.

Similarly, Resource Registry is used for registering available computing resources (cycles, memory, bandwidth, I/O capacity, etc.). In the case of FAFNER, a simple guideline is given to resource contributers so that sizes of the tasks (the length of the interval) given to them match well with their computers. A general periodic model of resource as a tuple of quantities is given in section 3. However, static, nominal resource availability often is a poor indicator on which to base resource management algorithms because the dynamic range of actual availability can be quite wide. Therefore dynamic instrumentation and performance monitoring would be enormously helpful in providing the more relevant resource availability to the resource manager.

Finally, the Resource Manager component contains the protocols and algorithms for allocation, scheduling, admission control, *etc.* In FAFNER, a hierarchy of task queues is used to distribute the tasks. In general, a particular concrete resource manager is responsible for a particular concrete model, but it may contain many different algorithms to manage the resource in that setting. It is worth noting that in this Framework, we allow the most general sort of resource manager which deals with dynamic tasks where multi-precision and multi-representation of application tasks are exploited as well as using dynamically instrumented, actual resource availability.

7 Test Application: WebSearch

Frameworks are most applicable to design challenges involving a *set of related applications* which are neither too nebulous (general-purpose computation) nor too restricted (single-instance software). To verify the applicability of RWC to a real-world application domain, we have identified *Web Search* as just such an appropriate application class. Web Search (specifically, image database search) offers (1) a broad literature of available techniques, and (2) a healthy number of publically available implementations of those techniques in library form.

These are important features, since we do not propose development of any new algorithms for content-based image database indexing and search. (An experimental prototype web image spider, centralized index, and web-based query engine interface have already been developed. The prototype implementation is in Perl, CGI, and C, and supports search of images based on overall color distribution.) Instead, the purpose of this sub-project is to use RWC software engineering techniques to refine the existing single-server prototype into a multiserver prototype, and then into a reusable application framework. This last refinement makes it possible to generalize and expand WebSearch to incorporate a wide range of well-established image compuation methods.

The WebSearch framework will consist of three top-level components, each of which may be multiply instantiated across a collection of Web servers: 1.) a C++ class hierarchy for dispatched image computation, 2.) a object-oriented database management system (OODBMS) component, and 3.) a graphical user interface (GUI) layer. Figure 4 shows a conceptual diagram of these layers.

The WebSearch framework is an extension of the Responsive Web Computing framework; WebSearch components will therefore inherit appropriate instrumentation and interfaces to interact with Resource Managers, Resource Registries, and Task Registries. This adds a fourth component to the first three: a RWC/WebSearch coordination system that

- selects among algorithms based on timeliness constraints and deadlines,
- balances tradeoffs between computation time and timing constraints (deadlines),
- dispatches image search and computation within the RWC,
- assembles the results for insertion in the OODBMS, and
- maintains the Web-published image index.

Most of the WebSearch research and development effort will focus on building the dispatched computation classes and the RWC coordination system. The OODBMS and GUI layers will consist mainly of software that is either available off-the-shelf or previously developed at Boston University.

Web Challenges The WWW is too large (and growing too fast) for a single web spider to crawl within a reasonable amount of time. Spiders for text-based search engines need to do this even though they fetch only text documents. Commercial engines like Lycos use a dozens of spider hosts. WebSearch picture agents will be fetching the potentially much larger image documents—in addition to performing computationally-intensive image statistics and other indexing information. Distribution of this web spider presents both a challenge and opportunity to test the responsive web computer framework.

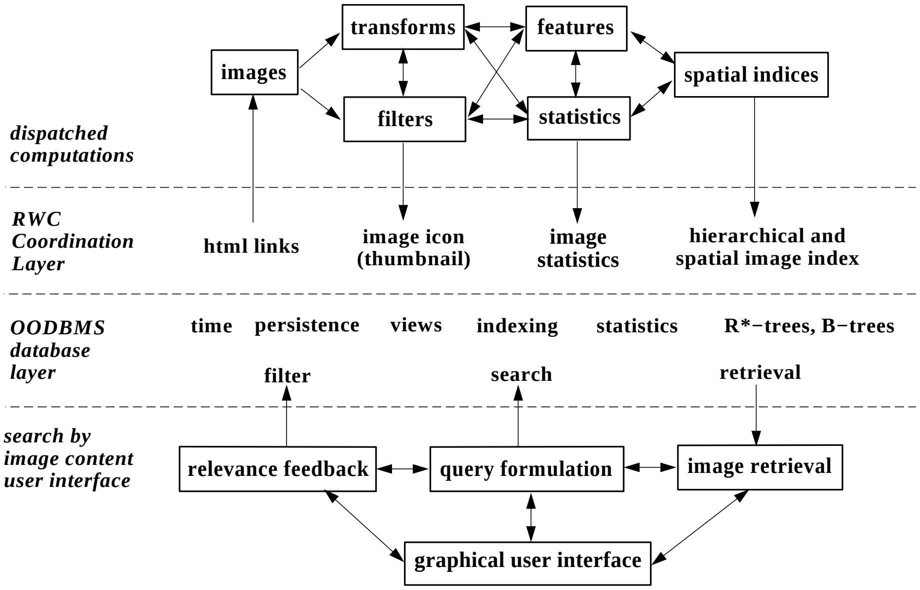

Figure 4: Diagram showing the four conceptual layers of the WebSearch application: 1.) a C++ class layer for dispatched image computation, 2.) a responsive web computing (RWC) coordination layer, 3.) a object-oriented database management system (OODBMS) layer, and 4.) a graphical user interface (GUI) layer. Most of the research and development effort will focus on building first two layers within the RWC framework. The OODBMS and GUI layer will consist mainly of software that is either available off-the-shelf or previously developed at Boston University.

7.1 C++ Class Layer for Dispatched Image Computation

The primary focus will be on the porting this prototype to the RWC. This requires respecification and implementation of the image crawler using C++ classes. We plan to leverage this effort by considering the ARPA Image Understanding Environment (IUE) classes and associated libraries. A preliminary version of this C++ class library is now freely available from AAI, Inc. The use of an established class specification should allow quick development, and enable leveraging image algorithms implemented by other researchers using the IUE.

An important focus of this effort will be to identify and develop a layer of classes generic to image representation and search. For instance, in the IUE class hierarchy it is known that certain spatial attributes and operations occur over and over [50]. An effort to abstract such properties into generic class layers should result in powerful generalizations that can be used in the Metacomputing Framework. The WebSearch implementation will start simple: using well-established histogram-based algorithms for search of images based on overall color similarity [89, 37, 39]. The specification of a generic layer will enable fast generalization to established texture-based algorithms [39, 33, 71, 90] and edge-orientation histogram algorithms [30, 32].

This generic layer will then be extended to include hierarchical and spatial representations for images. Abstraction of generic classes for spatial operations, attributes, and indexing will leverage work in the IUE. Established algorithms will be used for spatial subdivision (quad-trees, R*-trees) of images based on color, intensity, and texture properties [48, 73, 72, 88, 91]. In addition, rough figure-ground segmentation could be incorporated via clustering in conjunction with color and texture information [17, 38, 49, 58, 63]. This provides good "cut-outs" of foreground objects, and will allow search of images based on localized properties.

7.2 OO Database Layer (off-the-shelf)

Image thumbnails, links to web documents, and computed image index information are stored in a centralized object oriented database to support WebSearch. This functionality is administered by an object-oriented database layer using an off-the-shelf object oriented database manangement system (OODBMS) such as that of Illustra or Object Design Incorportated (ODI). Both OODBMSs provide the necessary support for search on spatial indexing (R*-trees), search using weighted image statistics, and HTML/WWW support. This makes it possible to leverage the WebSearch project using an "off-the-shelf" OODBMS.

7.3 Graphical User Interface

A web-based graphical user interface has already been developed as part of the WebSearch prototype system. It allows for query formulation, relevance feed-back, and image display using HTML and CGI-bin scripts. This GUI will constitute a graphical interface layer for the WebSearch system, and will be available at the Boston University Computer Science Department's web-site.

Two primary user interface paradigms will be available in the system. One allows users to define a search in terms of example images[18]: the user selects examples from a system-supplied menu, or the user creates a rough sketch example using a paint tool[45, 61]. The other paradigm allows users to select or circle a region of the image that is of interest, and then the system retrieves other images with similar regions in them. Both paradigms skirt problems associated with semantic labeling by letting users show pictures of what is desired. This also avoids the difficulties users may have in using words to describe what they are looking for. The resulting interface will be embodied in Java, CGI bins, and perl scripts. It will thus be available at a web site, allowing anyone with a Java-capable viewer to utilize the search engine for searching the web.

7.4 Algorithmic Families

The goal is develop a system that enables variable tradeoff between quality, computation speed, and accuracy of image computation. We have already begun to test such an approach in a prototype system for searching the Boston University WWW site for similar images based on color properties. We propose to extend this work so as to broaden the range of speed/accuracy available for searching on a particular image property, and to derive algorithms that have predictable performance with respect to one another.

The strategy will be to employ *algorithmic families* that subsume each other, balancing the trade-offs of speed vs. accuracy based on deadlines and other time constraints. The algorithmic family approach has already been demonstrated with color-based search in the WebSearch preliminary system. Histogram back-projection [89] is used to accurately search for images with overall similar color distributions. To achieve interactive but less accurate searches we use "average color" [37]. This imprecise but faster method makes it possible to efficiently filter through the whole image database, narrowing search to a manageable subset of candidate images for which the more expensive measures can then be computed. The filtered set is guarranteed to include as a subset the first N images considered most similar using the more expensive color histogram comparison [37]. Such a combination of measures makes it possible to satisfy the constraints of interactive search in the BU WebSearch prototype. This approach will be extended within the RWC framework.

8 Related Work

Distributed Computing, Internet Computing Various distributed computing software projects (Linda and Piranha [16], Phish [12]) have succeeded in linking the resources of a workstation cluster into a single computational surface. These efforts generally support some degree of dynamic configuration, allowing nodes to join and depart the computation in progress. The underlying model is still that of a centrally administered and scheduled computing resource, however, and existing NOW approaches to distributed computing do not scale to Internet-wide metacomputing.

One project which has made some progress toward breaking this barrier is Legion [34], which like its ancestor, Mentat, is a project at the University of Virginia focusing on managing internet resources for high performance computing. Some aspects of Legion's global metacomputing functionality (for example, their construction of a global Legion filespace atop the local Unix filesystems of the participating nodes) may in fact already be supplied by approaches which take advantage of the omnipresent World-Wide Web. Unlike Legion, however, our Responsive Web Computing project focuses on user-directed coordination of critical components. It sets the stage for wide-scale cooperative development, rather than mandating a single, centralized, general-purpose framework for coordination as in Legion.

The Harvest project [14] is another attempt to facilitate the location of data objects in the Internet and to speed up access to those objects. Like our project, it envisions a world-wide hardware base. However the

RWC goes beyond the scope of the Harvest system by combining distributed computation with distributed data objects, and through its emphasis on reliable computation with real-time guarantees.

Library and Framework Approaches Our use of frameworks, rather than simple libraries or layers, to organize the services and protocols offered by the RWC system, reflects the leading edge of a general trend in distributed computing. Tools and services that support a given problem domin don't stand alone — they are most easily integrated and optimized when they can take advantage of the application context they are called from. This additional information about the ways a set of classes cooperate to solve problems differentiates frameworks from more straightforward library approaches.

The POOMA framework,[52] for example, defines global and local service layers for the domain of particle physics — Matrix, Field, and Particle classes. This standard framework accelerates the development of physics codes, going beyond simple library support by providing a Parallel Abstraction Layer to mediate local and global views of distributed data. Like the RWC framework, POOMA demonstrates the process of capturing domain-specific expertise (about the coordination of all the distributed components which make up a complex problem domain) in a reusable form.

The framework approach also supports the integration of standard, off-the-shelf software components. When the "roles" in a distributed computation have been defined (by supplying the abstract class interfaces for each of the players), the concretization of those roles can consist of a suitably wrapped use of an existing tool or component. We will investigate the use of an off-the-shelf database manangement system (OODBMS) such as that of Illustra or Object Design Incorportated (ODI). Both OODBMSs provide the necessary support for search on spatial indexing (R*-trees), search using weighted image statistics, and HTML/WWW support. ARPA's own Image Understanding Environment project (currently under development at AAI, Inc.) will provide standard implementations and algorithms in support of WebSearch.

Resource Management for Responsiveness: Research in real-time scheduling was ushered by the pioneering Rate-Monotonic Scheduling work of Liu and Layland [57], which was followed by several projects that aimed to catalyze an improvement in the state of the practice for real-time systems engineering based on a solid, analytical foundation for real-time resource management. Examples include the Ada RTSIA and RMARTS Projects at the Software Engineering Institute [79], which led to several results in scheduling periodic tasks with synchronization requirements [78], mode change requirements [77], specified in Ada [76]. Another leading effort in scheduling periodic tasks was the introduction of the imprecise computation paradigm by Chih, Liu and Chung [21, 81]. Using that paradigm, rather than attempting to execute each task until completion—possibly missing the task's deadline—a trade-off is made with the quality of the result.

Research on scheduling both periodic and aperiodic (sporadic) tasks include the *resource reclaiming* [80] technique, which passively reclaims unused resources for the processing of other tasks when a task either executes less than its worst case computation time or when it is removed from the schedule. In a similar fashion, *slack stealing* algorithms [29, 93], which actively *steal* time from hard deadline periodic tasks and hard deadline aperiodic tasks in order to service aperiodic task requests.

Research on resource management for timeliness in distributed systems is limited to tightly-coupled systems. An example of such an effort is the Spring Kernel [85, 86], which includes support for admission control based on simple feasibility analysis, and includes techniques for distributed scheduling in a tightly-coupled multiprocessor system using random, focussed addressing, bidding and flexible algorithms [84].

There have been little work on scheduling to guarantee timeliness, even in the presence of failure [31]. An example of research work that tackles both real-time and fault-tolerance issues is the *primary/alternative* model [20, 51, 55, 60, 62], which is designed to deal with both timing and hardware failures in multiprocessor systems.

The task and resource specification paradigm we propose to use for our RWC lends itself naturally to previous work on *aperiodic servers*, whose sole purpose is to actively process aperiodic task requests. Examples include the Deferrable Server [87], Sporadic Server [82], Transient Server [70], Dynamic Priority Exchange, Total Bandwidth and Earliest Deadline Latest Servers [83]. Our proposed work is different because it considers not only CPU scheduling, but the scheduling and management of other resources as well (including memory and I/O) in a *loosely-coupled* large-scale distributed environment.

Generally speaking, most of the research outlined above for scheduling and resource management in real-time/fault-tolerant environments has concentrated on *"static"* systems with deterministic, homogeneous resources [31] for the purpose of satisfying rigid, safety-critical, *hard* deadlines. Such rigid requirements are incompatible with Metacomputing applications, which are subject to *soft*, or *firm* deadlines.

Our work on allowing Web computing resources to be shared is similar to the work on Virtual Private Resources (VPR) [74], which in turn is based on the Common Request Broker Architecture (CORBA) [36]. In VPR, *brokers* are allowed to act on behalf of clients to locate a server that would satisfy their service needs. However, in VPR there is no support for (or control over) the quality of services (such as meeting deadline, or protecting against failures) provided to clients.

Communication Infrastructure in the RWC: In the area of traffic analysis and nework support for the RWC, previous work is mainly distinguished by its assumption of much simpler traffic models. These simpler traffic models (Poisson or Markovian arrival processes) have affected the basic assumptions made in most previous network and protocol design. In contrast, our work is informed by the insight that traffic is self-similar; and the network models and transport protocols we propose are based on much more realistic assumptions about real traffic patterns.

For example, previous work in flow control protocols has emphasized an idealized, "steady-state" view of network traffic [46, 15]. In the steady-state approach to flow control, it is assumed that the objective of the flow control protocol is to find the amount of transmission bandwidth available along a path, and to adjust transmission rate to match that available bandwidth. Such algorithms use, *e.g.,* time-averaged statistics to measure round-trip time on a path and throughput being achieved on a path[44, 22].

Image Search Application: For some time now there have been "spiders" crawling the World Wide Web, collecting index information about the text documents they find. These search engines extract text indexing information that is later used to guide interactive searches without having to retrieve the actual web documents. The scale of these databases is impressive. For instance, Lycos, a web search engine at CMU, supports interactive keyword-based searches for an index of millions of web documents. To remain up to date, the engine retrieves and analyzes thousands of documents daily. What is needed is an equivalent web image search engine that crawls the web collecting information about the images it finds, computes the appropriate image decompositions and indices, and stores this extracted information for Lycos-style searches based on image content.

General approaches to image search provide an arsenal of image decompositions and discriminants that can be precomputed for images: color histograms [37, 39, 89], edge orientation histograms [30, 32] texture measures [39, 33, 71, 90], shape invariants [92] or shape encodings [19, 35, 47], eigendecompositions [75, 94], wavelet decompositions [45], etc. At search time, users can select a weighted subset of these decompositions to be employed for computing image similarity measurements during retrieval. We can draw on this extensive background of algorithms and implementations to build our image search framework, letting us focus our attention on the design of the distributed image search system.

Currently there are a few WWW sites that offer query by image content for a local database: UltiMedia (IBM Almaden), VIS (Virage), Cypress (Berkeley), Photobook (MIT), VisualSEEK (Columbia), Jacob (U. Palermo), WebSearch (Boston U.). Each site provides query-by-example search for a on-site, centralized repository that contains a relatively small number of images (on the order of 10K images or less). Currently no site is available that can provide image-based search of web documents outside the central server. The proposed effort to use the RWC framework will enable coordinated searches beyond the confines of an on-site database.

Most of these query by image content systems assiduously avoid the localized image comparisons. Some systems (e.g., IBM Ultimedia) assume a human operator will assist by circling regions of interest during the database building phase. This will be unacceptable in building image agents for the RWC, where there will be literally millions of images in the database. Recently there has been work done in at MIT to automatically detect faces and to compute figure/ground segmentation [67]. More approaches to search on localized properties are needed. Part of the proposed effort addresses this concern using well-established quad-tree methods[48, 73, 72, 88, 91].

9 Plan and Expected Results

The Web has not only opened up new approaches to wide-area high performance computation, but new strategies for propagating experimental results as well. Immediate web-publication of hyperlinked software and documentation brings previously arcane back-room processes — project management, software engineering, and phased technology transfer — into the daylight.

We have designed the Responsive Web Computing project as a showcase for high-visibility technology transfer — a Web-based methodology for transforming algorithms into services, services into a reusable design framework, and framework-based prototypes into usable Web-based HPC applications.

The following sections describe the concepts and software which will emerge from this project in a reusable online form, in three tiers:

- **RWC Techniques:** fundamental algorithm and protocol improvements.
- **RWC Services:** middleware for timeliness and fault-tolerance.
- **RWC Framework:** protocols and interfaces that synthesize a unified distributed design environment from available services, libraries, and toolkits for RWC and end-user applications.

D1. RWC Techniques.

GDIS Traffic Analysis. Assess the self-similarity of traffic in globally distributed information systems (GDIS), in three stages:

- Study the relationship between the heavy-tailed distribution of file sizes and their originating environment: general purpose, academic, business, scientific, embedded, and real-time computing.
- Identify the precise levels of self-similarity to be expected in GDIS through simulation of the effects of network protocols, caching, and user preferences.
- Finally, develop a model of GDIS network traffic parameterized by system type, transport model, and network bandwidth and delay characteristics.

Transport Protocol Layer.

- Design a pallette of TCP/IP implementations suitable for various application needs (*e.g.*, real-time traffic) and infrastructure constraints (*e.g.*, mobile computing).
- Specify a new "greedy" flow-control regime for TCP in the presence of self-similar traffic, anticipating high variation in competing traffic demand on a shared link.

Wave Protocol Design. Wave piggy-backs its operation over routing in order to leverage its name service and route-selection functionality, for the purpose of distributed caching for load-balancing.

- Protocol: mechanisms for deciding when to make a copy and to whom to send it, as well as locating a copy from which a request can be served.
- Design a number of policy heursitics, such as selection rule for documents to push to a child or release to a parent, or the method for aging gossiped server loads, *etc.* These heuristics represent the means by which the protocol can be tuned in operation to achieve its goals under widely varying conditions.
- Derive load balancing requirements; servers along each path of the tree should be approximately load-balanced, with no severe breakdowns in the long-term.

Evaluating Load-balancing and Stability. Strictly global load-balance is not within realistic reach, so we concentrate on load-balance over client-server paths. However, achieving load-balance on average is not sufficient if severe breakdowns can occur in the long run, hence the stability requirement.

- Extend MaRS (Maryland Routing Simulator) to model Wave running in conjunction with routing.
- Simulate an *Internet patch* by modeling cross-traffic originating from, or destinted to, nodes outside of the patch being simulated.
- Implement Wave on top of extended MaRS.
- Measure extent of load-balance per tree path, and globally over the simulated network patch.
- Evaluate the long-term stability of load-balance.

D2. RWC Services.

To turn the RWC techniques into reusable components, three sets of services must be implemented or reused from existing toolkits and software repositories.

Responsive Computing Services.

- Implement *Bprobe* and *Cprobe,* which measure raw or link bandwidth and currently available bandwidth between clients and servers.
- Design information dissemination, speculative document service, and caching services that can be initiated by multiple *cooperative servers.*
- Extend HTTP servers to allow middleware services to convey *"hints"*—metainformation about prefetching and dissemination—to clients.
- Implement resource management services that allow the assignment of computations to RWC resources in a way that promotes predictable, responsive performance.

Web Computing Services. These services will be responsible for managing all stages of the metacomputing lifecycle:

- software distribution, software update, administrative control, etc.
- task allocation, algorithm execution, the accumulation and presentation of its computational results
- demonstrate a suite of concrete services for the focus application (WebSearch)
  - distributed database indexing
  - distributed database query construction and dispatch
  - collective database performance monitoring and feedback
  - data migration (to resolve "hot spots" in the distributed database)
  - merge and ordering of search results
  - user interface presentation
  - interactive search refinement

D3. RWC Framework.

We observe a need for two flavors of framework support in the RWC project:

- **Responsive Web Computing** supplies new services and relationships among service providers to improve predictability and throughput of Web computation.
- **WebSearch Framework** builds on the Responsive Web Computing framework to verify applicability to widely distributed search problems.

The union of these two support systems we call, simply, the RWC Framework. These are the primary RWCF development tasks common to both phases:

- extract design patterns which may be appropriate to the larger application class.

  These can be new peer-to-peer coordination strategies or additional layers for real-time and reliability guarantees (for Responsive Web Computing), or different image search algorithms (for WebSearch).
- publish these patterns (and the image database software and the underlying resource manager that concretizes the patterns), on the WWW
- build incremental-collaboration tools to allow new image search application to be instantiated as collaborations of database sites, starting with the software from the prototype application, but varying one or more of the specific Web computing and resource management strategies.

Responsive Web Computing Framework The basic Web computing framework will provide management support for Web-distributed applications, emphasizing basic network management functionality. The framework then goes on to define an additional set of cooperating instrumentation, monitoring, and resource management services for applications which require improved predictability and reliability.

- standardize interfaces between application programs and the RWC resource manager, and define the interactions within a collection of RWC services — caching, replication, prefetching, performance prediction and synchronization — and RWC middleware
- specify the roles and protocols in the RWC framework: the Resource Management Interface, a Resource Registry, a Task Registry, and a Resource Manager.
- Identify strategies for making the resource manager aware of varying requirements of different algorithmic components of the services, in order to best match available resources with the computing tasks that use these services.

Web Search Framework. The development of the Web Search Framework will progress in two dimensions: engineering form versus algorithmic content. In the first dimension, we will

- transform a centralized application (eg. the WebSearch Engine for a single server) to a one-shot distributed application using the WebSearch Engine services listed above, and then
- generalize this distributed application to a more reusable, object-oriented framework (the distributed WebSearch Framework).

The second dimension (algorithmic content) moves from simple placeholder algorithms and their code prototypes (Build 1) to a collection of releasable software based on the best available variants of those algorithms (Build 3). This plan allows us to proceed with a prototype Web-distributed image search application while the “real one” is still being worked on.

- Define the set of cooperating services which make up the Web Search application and their top-level interaction (abstract interfaces)
- Implement prototypes of the specific functionality of each of these roles in the framework, or rewrap off-the-shelf classes. For example, we may draw upon standard libraries provided by the ARPA Image Understanding Environment by AAI, Inc., or existing OODBMS and GUI classes.

- Revise and adapt these implementations by linking new code for each role as they become available.
- Generalize and expand the framework in this way to incorporate a wide-range of both well-established and novel image compuation methods.

Summary: Technology Transfer Path

Such a distributed image search framework for globally-distributed image databases has direct application in areas of national strategic importance such as defense tactical image analysis, national security, intelligence gathering, and crime prevention.

To make both the image search engine and the Responsive Web Computing System available to the research community, we will publish the developing frameworks and ongoing concretization of their components over the WWW. Incremental framework development tools and graphical user interface tools from Cooperating Systems Corporation will allow RWCF users to visualize and monitor all the distributed players that fill the roles of the abstract frameworks, and provide point-and-click version control over the distributed software base.

References

[1] M. Ahamad, G. Neiger, J.E. Burns, P. Kohli, and P.W. Hutto. Causal memory: definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995.

[2] Azer Bestavros. IDA-based disk arrays. Technical Memorandum 45312-890707-01TM, AT&T, Bell Laboratories, Department 45312, Holmdel, NJ, July 1989.

[3] Azer Bestavros. AIDA-based Communication for Distributed Real-Time Applications. In *Proceedings of the Second IEEE Network Management and Control Workshop*, Tarrytown, NY, September 1993.

[4] Azer Bestavros. An adaptive information dispersal algorithm for time-critical reliable communication. In Ivan Frisch, Manu Malek, and Shivendra Panwar, editors, *Network Management and Control, Volume II.* Plenum Publishing Corporation, New York, New York, 1994.

[5] Azer Bestavros. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proceedings of SPDP'95: The 7^{th} IEEE Symposium on Parallel and Distributed Processing*, San Anotonio, Texas, October 1995.

[6] Azer Bestavros. Using speculation to reduce server load and service time on the www. In *Proceedings of CIKM'95: The 4^{th} ACM International Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1995.

[7] Azer Bestavros. Aida-based real-time fault-tolerant broadcast disks. Technical Report TR-96-001, Boston University, CS Dept, Boston, MA 02215, January 1996.

[8] Azer Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering*, New Orleans, Louisiana, March 1996.

[9] Azer Bestavros, Robert Carter, Mark Crovella, Carlos Cunha, Abdelsalam Heddaya, and Sulaiman Mirdad. Application level document caching in the internet. In *IEEE SDNE'96: The Second International Workshop on Services in Distributed and Networked Environments*, Whistler, British Columbia, June 1995.

[10] Azer Bestavros and Carlos Cunha. A prefetching protocol using client speculation for the www. Technical Report TR-95-011, Boston University, CS Dept, Boston, MA 02215, April 1995.

[11] Azer Bestavros and Dimitrios Spartiotis. Probabilistic Job Scheduling for Distributed Real-time Applications. In *Proceedings of the First IEEE Workshop on Real-Time Applications*, New York, NY, May 1993.

[12] Robert D. Blumofe and David S. Park. Scheduling large-scale parallel computations on networks of workstations. In *Proceedings of the Third International Symposium on High-Performance Distributed Computing*, pages 96–105, San Francisco, California, August 1994.

[13] Jean-Chrysostome Bolot. Characterizing end-to-end packet delay and loss in the internet. *Journal of High Speed Networks*, (2):305–323, 1993.

[14] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, Colorado, August 1994.

[15] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of SIGCOMM '94*, pages 24–35, 1994.

[16] N. Carriero and D. Gelernter. Linda in context. *Comm. ACM*, April 1989.

[17] M. Celenk. A color clustering technique for image segmentation. *CVGIP*, 52:145–170, 1990.

[18] N. Chang and K. Fu. Query-by-pictorial-example. *IEEE Transactions on Software Engineering*, SE-6(6):519–524, 1980.

[19] Z. Chen and S. Y. Ho. Computer vision for robust 3D aircraft recognition with fast library search. *Pattern Recognition*, 24(5):375–390, 1991.

[20] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.

[21] Jen-Yao Chung, Jane Liu, and Kwei-Jay Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transaction on Computers*, 19(9):1156–1173, September 1990.

[22] David D. Clark. Window and acknowledgement strategy in TCP. Request for Comments 813, July 1982.

[23] D.D. Clark and D.L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proc. ACM SIGCOMM '90*, pages 200–208, Sep. 1990. Published as special issue of Computer Communication Review, vol. 20, number 4.

[24] RSA130 Collaboration. FAFNER: Factoring via Network-Enabled Recursion. Available from `http://cooperate.com/cgi-bin/FAFNER/factor.pl`.

[25] Mark Crovella and Azer Bestavros. Explaining world wide web traffic self-similarity. Technical Report TR-95-015, Boston University, CS Dept, Boston, MA 02215, August 1995.

[26] Mark Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of SIGMETRICS '96*, page (to appear), 1996.

[27] Mark E. Crovella and Robert L. Carter. Dynamic server selection in the internet. In *Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95)*, Mystic, CT, August 1995.

[28] Carlos Cunha, Azer Bestavros, and Mark Crovella. Characteristics of www client-based traces. Technical Report TR-95-010, Boston University, CS Dept, Boston, MA 02215, April 1995.

[29] R. I. Davis, K. W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 222 – 231, December 1993.

[30] A. Evans, N. Thacker, and J. Mayhew. The use of geometric histograms for model-based object recogntion. In *Proc. BMVC*, 1993.

[31] Donald Fussell and Miroslaw Malek, editors. *Responsive Computer Systems: Steps toward fault-tolerant real-time systems.* Kluwer Academic Publishers, Norwell, Massachusetts, 1995.

[32] S. Gallant and M. Johnston. Image retrieval using image context vectors: First results. In *Proc. SPIE Conf. on Storage and Retrieval of Image and Video Databases III*, San Jose, CA, February 1995.

[33] M. Gorkani and R. Picard. Texture orientation for sorting photos at a glance. In *Proc. IEEE Conf. on Pattern Recogntion*, 1994.

[34] Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Jr. Paul F. Reynolds. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, University of Virginia, June 1994.

[35] W. Grosky, P. Neo, and R. Mehrotra. A pictorial index mechanism for model-based matching. *Data and Knowledge Engineering*, 8:309–327, 1992.

[36] Object Management Group. Common object request broker architecture, 1991.

[37] J. Hafner, Harpreet Sawney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE T-PAMI*, 1(7):729–736, 1995.

[38] G. Healey. Segmenting images using normalized color. *IEEE Trans. on Systems, Man, and Cybernetics*, 22:64–73, 1992.

[39] G. Healey and D. Slater. Global color constancy: recognition of objects by use of illumination-invariant properties of color distributions. *JOSA-A*, 11(11):3003–3010, 1994.

[40] A. Heddaya and K. Park. Mapping parallel iterative algorithms onto workstation networks. In *Proc. 3rd IEEE International Symposium on High Performance Distributed Computing, San Francisco*, pages 211–218, Aug. 1994.

[41] A. Heddaya, K. Park, and H.S. Sinha. Using warp to control network contention in Mermera. In *Proc. 27th Hawaii International Conference on System Sciences, Maui, Hawaii*, pages 96–105, Jan. 1994.

[42] A. Heddaya and H.S. Sinha. An overview of MERMERA: a system and formalism for non-coherent distributed parallel memory. In *Proc. 26th Hawaii International Conference on System Sciences, Maui, Hawaii*, pages 164–173, Jan. 5–8 1993.

[43] A. Heddaya and H.S. Sinha. Distributed parallel computing in mermera: Mixing noncoherent shared memories. Technical Report BU-CS-96-005, Boston University, Computer Sciene Dept., Mar. 1996.

[44] Information Sciences Institute. Transmission control protocol. Request for Comments 793, available from `http://www.isi.edu/in-notes/std/std7.txt`, September 1981.

[45] C. E. Jacobs, A. Finkelstein, and D. Salesin. Fast resolution image querying. In *SIGGRAPH Conference Proceedings*, pages 277–286, 1995.

[46] Van Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, pages 314–329, 1988.

[47] H. V. Jagadish. A retrieval technique for similar shapes. In *Proc. International Conference on Management of Data, ACM SIGMOD 91*, pages 208–217, Denver, CO, May 1991.

[48] A. Klinger and C.R. Dyer. Experiments on picture representation using regular decomposition. *Computer Graphics and Image Processing*, 5:68–105, 1976.

[49] G. J. Klinker, S.A. Shafer, and T. Kanade. A physical approach to color image understanding. *IJCV*, 4:7–38, 1990.

[50] C. Kohl, J. Hunter, and C. Loiselle. Towards a unified iu environment: Coordination ofd existing iu tools with the iue. In *Proc. ICCV 95*, 1995.

[51] C. M. Krishna and K. G. Shin. On scheduling tasks with a quick recovery from failure. *IEEE Transactions on Computers*, 35(5):448–455, May 1986.

[52] LANL Advanced Computing Laboratory. The POOMA Framework. Available from `http://www.acl.lanl.gov-/PoomaFramework`.

[53] W. E. Leland and D. V. Wilson. High time-resolution measurement and analysis of LAN traffic: Implications for LAN interconnection. In *Proceeedings of IEEE Infocomm '91*, pages 1360–1366, Bal Harbour, FL, 1991.

[54] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2:1–15, 1994.

[55] A. Liestman and R. Campbell. A fault-tolerant scheduling problem. *IEEE Transaction on Software Engineering*, SE-12(11):1089–1095, November 1986.

[56] R.J. Lipton and J.S. Sandberg. PRAM: a scalable shared memory. Technical Report CS-TR-180-88, Princeton University, Sep. 1988.

[57] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environments. *Journal of the Assocation of Computing Machinery*, 20(1):46–61, January 1973.

[58] J.Q. Liu and Y.H. Yang. Multiresolution color image segmentation. *IEEE T-PAMI*, 16:689–700, 1994.

[59] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freedman and Co., New York, 1983.

[60] D. Mosse, R. Melhem, and S. Ghosh. Analysis of a fault-tolerant multiprocessor scheduling algorithm. *IEEE Fault Tolerant Computing*, pages 16–25, 1994.

[61] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture, and shape. In *Proc. SPIE Conf. on Storage and Retrieval of Image and Video Databases*, volume 1908, February 1993.

[62] Y. Oh and S. Son. An algorithm for real-time fault-tolerant scheduling in multiprocessor systems. In *Fourth Euromicro Workshop on Real-time Systems*, 1992.

[63] Y. Ohta, T. Kanade, and T. Sakai. Color information for region segmentation. *CVGIP*, 13:222–241, 1980.

[64] Kihong Park, Gi Tae Kim, and Mark E. Crovella. The effects of traffic self-similarity on tcp performance. Technical report, Boston University Computer Science Department, 1996.

[65] Kihong Park, Gi Tae Kim, and Mark E. Crovella. On the cause and effect of self-similar network traffic. Technical report, Boston University Computer Science Department, 1996.

[66] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of poisson modeling. In *Proceedings of SIGCOMM '94*, 1994.

[67] A. Pentland, R. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. *IJCV*, to appear.

[68] Barry M. Leiner (program chair). Workshop on Middleware (in conjunction with sigcomm'95), March 1995. Cambridge, MA.

[69] Michael O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, April 1989.

[70] S. Ramos-Thuel and J. K. Strosnider. The transient server approach to scheduling time-critical recovery operations. In *Real-Time Systems Symposium*, pages 286 – 295, December 1991.

[71] R. Rao and B. Schunck. Computing oriented testure fields. *CVGIP: Graphical Models and Image Processing*, 53(2):157–185, 1991.

[72] H. Samet. Applications of spatial data structures. In *Addison-Wesley*, 1990.

[73] H. Samet, A. Rosenfeld, C.A. Shaffer, and R.E. Webber. A geographic information system using quadtrees. *Pattern Recognition*, 17:647–656, 1984.

[74] Thomas Schwotzer and Thomas PreuB. Towards the guarantee of service parameters, 1996. Internal Report.

[75] S. Sclaroff and A. Pentland. Modal Matching for Correspondence and Recognition. *IEEE T-PAMI*, 17(6):545–561, 1995.

[76] L. Sha and J. Goodenough. Real-time scheduling theory and ADA. *IEEE Computer*, April 1990.

[77] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1, 1989.

[78] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 9 1990.

[79] Lui Sha, Mark Klein, and John Goodenough. Rate monotonic analysis for real-time systems. In Sang Son, editor, *Foundations of Real-Time Computing: Scheduling and Resource Management.* 1994. Also available as CS-RT-3181 from the University of Maryland.

[80] C. Shen, K. Ramamritham, and J. A. Stankovic. Resource reclaiming in real-time. In *Real-Time Systems Symposium*, pages 41 – 50, December 1989.

[81] Wei-Kuan Shih, Jane Liu, and Jen-Yao Chung. Algorithms for scheduling imprecise computations with timing constraints. *SIAM journal of Computing*, July 1991.

[82] B. Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1:27–60, 1989.

[83] Marco Spuri and Giorgio C. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 2 – 11, December 1994.

[84] John Stankovic. Decentralized decision making for task allocation in a hard real-time system. *IEEE Transactions on Computers*, March 1989.

[85] John Stankovic and Krithi Ramamritham. The design of the Spring kernel. In *Proceedings of the Real-Time Systems Symposium*, pages 146–157. IEEE Computer Society Press, December 1987.

[86] John Stankovic, Krithi Ramamritham, and S. Cheng. The Spring Kernel: A new paradigm for real-time systems. *IEEE Software*, pages 54–71, May 1992.

[87] Jay Strosnider. *Highly Responsive real-time token rings.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, August 1988.

[88] G.J. Sullivan and R.L. Baker. Efficient quadtree coding of images and video. *Image Processing*, 3(3):327–331, 1994.

[89] M. Swain and D. Ballard. Color indexing. *IJCV*, 7(1):11–32, 1991.

[90] H. Tamura, S. Mori, and T. Yamawaki. Textural features corresponding to visual perception. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(6):460–472, 1978.

[91] S.L. Tanimoto and T. Pavlidis. A hierarchical data structure for picture processing. *CGIP*, 4:104–113.

[92] G. Taubin and D. Cooper. Recognition and positioning of rigid objects. In *Proc. SPIE Conf. on Geometric Methods in Computer Vision*, volume 1570, 1991.

[93] S. R. Thuel and J. P. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Real-Time Systems Symposium*, pages 22 – 33, December 1994.

[94] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[95] L.G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33(8):103–111, Aug. 1990.

[96] P. Wang and W.E. Weihl. Scalable concurrent B-trees using multi-version memory. *J. Parallel and Distributed Computing*, 32(1):28–48, Jan. 1996.