

PUNCH: An architecture for Web-enabled wide-area network-computing

Nirav H. Kapadia José A. B. Fortes

School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA

This paper addresses the architectural issues that arise in the design of a universally accessible wide-area network-computing system that is capable of making automatic cost/performance tradeoff decisions at run-time. The core system is designed around a three-level hierarchically distributed architecture, a choice driven by the dynamic, incremental, and distributed nature of the information associated with run-time cost/performance tradeoff decisions. Support for independent replication of each component in the hierarchy contributes to the overall scalability and reliability of the architecture. Meta-information is managed in a scalable manner by employing self-encoded resource identifiers that allow $O(1)$ access to all managed information. Security and access control across administrative domains are provided by partitioning the infrastructure into independently-managed cells, and by giving administrators the ability to customize user-views directly at the location at which the request is processed. Demand-driven resource management is achieved by predicting the run-specific resource usage characteristics of tools via machine learning techniques. The concepts described in this paper are embodied in the Purdue University Network-Computing Hubs (PUNCH), a demand-based network-computing system that allows users to access and run *unmodified* tools via standard world-wide web browsers. Tools do not have to be written in any particular language, and access to source or object code is not required. The PUNCH infrastructure can be distributed in a manner that allows tools to be (user-transparently) executed wherever they reside. Currently, PUNCH contains over thirty tools developed by eight universities and four vendors, and serves more than five hundred users. During the past three years, PUNCH users have logged more than one million hits and have performed over seventy thousand simulations.

1. Introduction

There is increasing evidence to support the view that, in the future, computing will be network-based and service-oriented. This view implicitly assumes the existence of an underlying infrastructure capable of supporting network-accessible, *demand-based computing*. A demand-based computing system can be characterized by its universal accessibility and its ability to make automatic cost/performance tradeoff decisions at run-time. Universal accessibility can be provided by leveraging a widely-used networked infrastructure such as the world-wide web [31]. Cost/performance tradeoff decisions require that the computing system be able to decide how (which implementation - e.g., sequential versus parallel) and where (which platform) to execute a tool. This paper describes the architectural issues involved in the design of a demand-based network-computing system, and frames the discussion within the context of a widely used system (the Purdue University Network-Computing Hubs, or PUNCH) that allows users to access and run *unmodified* software tools via standard world-wide web browsers. PUNCH can be accessed at “<http://www.ecn.purdue.edu/labs/punch/>”. Courtesy accounts are available.

On-going research on network-computing can be divided into two categories. The focus of work in the first category is to provide application programmers with the tools required to (easily) develop applications that are capable of accessing and using globally distributed resources, with

an emphasis on the ability to scale each application to a global level. Examples of such work include ATLAS [5], Globe [21,43], Globus/GUSTO [15,16], IceT [44], Legion [17–19], ParaWeb [8], and Symera (originally named Symbio) [24]. Work in the second category is geared towards providing *end users* with the means to transparently access and use globally distributed (software and hardware) resources. Examples of such work include CCS [36], MMM [20], MOL [37], NetSolve [10], Ninf [40,41], PUNCH [28,29], RCS [2,3], Rivendell [14,25], Schooner [22,23], VNC [39], and WinFrame [11]. Although the systems in the two categories currently address different issues and are (largely) developed independently, they complement each other in the sense that, together, they will enable true wide-area distributed computing (the “computational grids” of [16]).

The work described in this paper belongs to the second category. It was motivated by the fact that many of the systems and technologies that currently allow computing on the web target a single or a relatively small set of tools and/or work within controlled environments in which many of the issues that arise in production environments can be ignored.

Solutions that target individual tools tend to be non-reusable in spite of the fact that they involve a significant amount of duplicated effort.¹ For example, a large number of systems (e.g., the Exploratorium [1], JSPICE [42],

¹ Reusability, in this context, implies an ability to reuse the computing system with other applications without making any modifications to the system itself.

and others) are based on scripts that need to be modified in order to add any new application to the system. Other designs are more flexible. The MOL prototype, for example, employs static web interfaces that can be adapted for individual tools. The NetSolve, Ninf, and RCS systems are based on structured designs that target numerical software libraries. However, static interfaces are not adequate for all tools, and structured approaches cannot be easily applied to general-purpose applications. Another problem is that the majority of these designs assume the availability of the source and/or object code for the applications – which effectively precludes the installation of most commercial tools.

Solutions that address individual issues are generally reusable, but the tasks of adapting them for a production environment and integrating them into a complete computing infrastructure are non-trivial. For example, VNC and WinFrame provide very flexible mechanisms for exporting graphical displays to remote consoles in a platform-independent manner – but, by themselves, the technologies do not help address other issues (e.g., access control and management) that arise in a wide-area distributed computing environment.

The goal of the PUNCH project is to design and deploy a production wide-area computing framework in which the computing infrastructure is not tied to the characteristics of individual applications. Functionally, this is equivalent to designing a multi-user *operating system* for networked resources that provides user-transparent file, process, and resource management functions, handles security and access control across multiple administrative domains, and manages state information (session management). The need for such a system and the associated benefits in the context of the world wide web have also been recognized in [33,38].

A network-computing system can be characterized in terms of: 1) its interface to the external world, 2) its internal architecture, 3) the class of software resources that it can support, and 4) the capabilities of its resource management framework. This paper elaborates on the issues involved in the design of the internal architecture of a universally accessible, demand-based network-computing system. The paper is organized as follows. Section 2 outlines the issues that drive the architecture selection process. Section 3 describes the manner in which the basic ideas discussed in Section 2 are embodied in the PUNCH infrastructure, along with an empirical evaluation of specific aspects of the infrastructure. Finally, Section 4 presents concluding remarks, and Section 5 outlines the ongoing work associated with the PUNCH project.

2. Design Issues

The issues that drive the design of a demand-based network-computing system are shown in Figure 1. The focus of this paper is on the first four sets of issues shown in the figure.

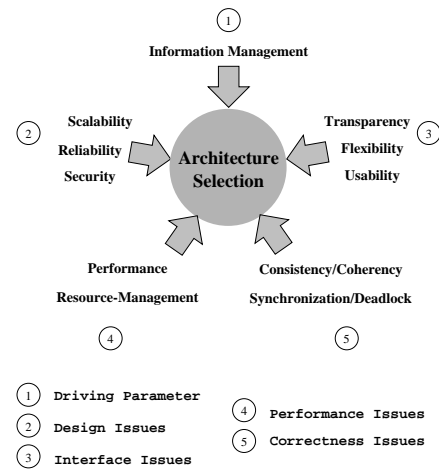


Figure 1. The architecture-selection process.

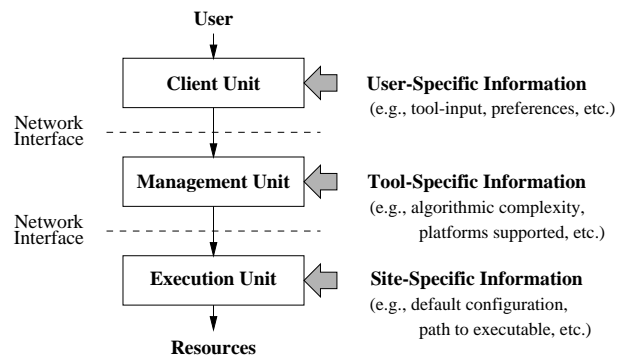


Figure 2. Core system architecture.

2.1. Information Management

The tool-specific cost/performance tradeoff decisions required for demand-based computing are primarily based on two factors: 1) the portability of the given tool (e.g., the platforms on which it is supported), and 2) its run-specific resource-usage characteristics in terms of CPU time, network data-transfer time, memory usage, and disk-space requirements. The decisions are also affected by site-specific policies and configurations; this is especially true for systems that span multiple administrative domains.

The architectural design of a demand-based computing system is implicitly tied to the nature of the information associated with the factors described above. The information is incremental in the sense that it has to be constructed from pieces available at different points in time. For example, while portability information is usually available a priori, resource usage is generally tied to the input supplied to the tool at run-time. Also, the information is often defined by independent sources at (potentially) different locations. For example, the run-specific information is obtained from users, whereas the site-specific information is specified by administrators. The tool-specific information is implicitly specified by the developers of the tool, and is generally independent of the location at which the tool is used.

The dynamic, incremental, and distributed nature of the

information associated with cost/performance tradeoff decisions makes it extremely difficult to collect and maintain the information at a centralized location in a scalable manner. Thus, it becomes necessary to distribute components of the network-computing system in a manner that allows the information to be used *in situ*. This can be accomplished by using a *vertically distributed* architecture. Vertical distribution involves a hierarchy of processing nodes with differing functionalities – all transactions enter and leave the system at the lowest level and pass through all levels [35]. It is necessary to keep the number of levels in such a system down to a minimum because the overhead of processing any given request increases with the number of levels.

Network-computing systems provide a wide range of services ranging from data management and directory information to tool invocation. Because of the diverse demands imposed by the transactions associated with these services, reliability and performance are improved by employing a particular form of vertical distribution in which each level possesses a degree of self-sufficiency – all transactions still enter and leave the system at the lowest level, but all requests do not have to pass through all levels of the architecture [35]. The hierarchically distributed architecture derived from these constraints is shown in Figure 2.

2.2. System Architecture

The core system architecture is designed around a three-level, message-passing hierarchy consisting of client, management, and execution units (see Figure 2). In the context of this architecture, a network-computing system is defined as a network-accessible collection of one or more instances of these components. Instances of a given component are not required to provide identical services (e.g., two management units may support different sets of tools). Also, for overall scalability, the basic three-level hierarchy is extended by allowing management units to forward requests to other management units (all requests for tool execution are eventually forwarded to execution units).

Client units serve as a conduit for user-specified commands, preferences, and directives, and facilitate the transfer of tool-related input and output. In general, this component will be integrated into the local computing environment (in a UNIX shell, say) or in a front-end desktop infrastructure (as in PUNCH, for example; see Section 3), allowing users to transparently access the network-computing system. Consequently, it is essential to design the client unit in a manner that allows it to be reused in a wide variety of environments (ideally, it should be architecture-neutral). This can be best achieved by adopting a thin-client approach; the client unit should forward user-requests to a management unit after minimal syntax and error checking.

Management units primarily act as demand-driven scheduling engines for associated software and hardware resources (see Figure 3). They also enforce resource-specific access control policies (e.g., licensing and ownership concerns). Functionally, a management unit must analyze (i.e., pre-

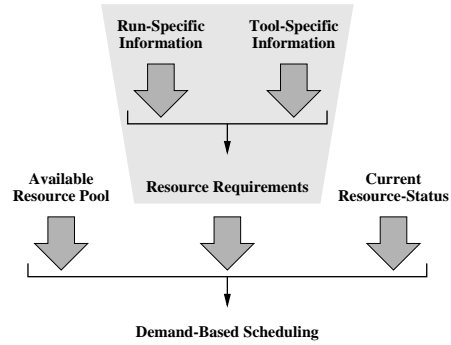


Figure 3. Demand-based scheduling.

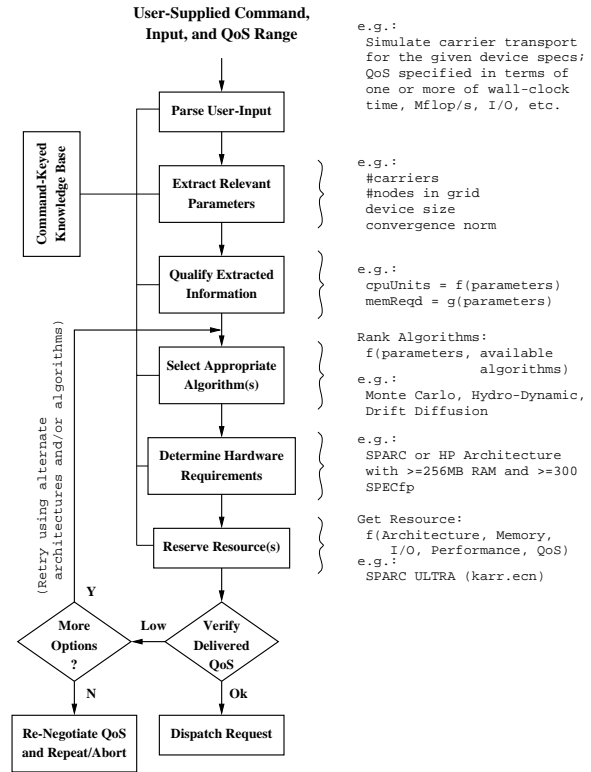


Figure 4. Management unit functionality.

dict) the requirements of a given request and then match the requirements with appropriate resource(s) from those available to the user initiating the request. The resource requirements for a given request can be predicted in a user-transparent manner by qualifying tool-specific resource usage characteristics with run-specific tool input supplied by the user (the approach used in PUNCH is described in Section 3.6). Once the requirements are known, an applicable set of implementations (e.g., sequential and parallel) and platforms (e.g., SPARC-5 and HP-9000/C110) can be determined. At this point, the specific implementation (i.e., executable) and the execution unit to be used for the run can be explicitly evaluated on the basis of applicable scheduling policies, and the request can be forwarded to the selected execution unit. The steps involved in this process are shown in Figure 4.

Management units primarily deal with information associated with tool-specific resource usage characteristics and institution-specific access control policies. This information is best maintained at a few strategic sites within a given organization. Management units can then be co-located at such sites, subject to scalability and reliability constraints. Given this distribution of (a relatively few) management units at key locations, it is desirable to incorporate the bulk of the resource management functionality of the network-computing system into this level of the hierarchy; the resulting system will be easier to maintain and upgrade.

Execution units present management units with a consistent, parameterized interface to hardware resources. The resources may consist of individual machines or locally managed sub-systems (e.g., a network of workstations managed by Condor [34]). When managing an individual machine, the execution unit simply initiates runs and handles data-transfer on behalf of authorized management units. When serving as an interface to locally managed sub-systems, the execution unit: 1) translates instructions² from authorized management units to a format that is suitable for the local resource-management system, and 2) facilitates the negotiation of quality-of-service parameters. Execution units also keep management units apprised of any unforeseen situations that may affect the execution of a request (the management unit could then re-allocate the run or initiate a task-specific error management procedure).

2.3. Scalability and Reliability

The scalability of a network-computing system can be defined in terms of its ability to continue to perform well with an increasing number of: 1) users, 2) software resources (tools), 3) hardware resources, and 4) administrative domains. This section discusses the scalability and reliability of the architecture with respect to each of these factors.

Scalability with respect to users can be achieved as follows. The top (client unit) level of the hierarchy is implicitly scalable when client units are integrated into the local computing infrastructure (e.g., in a UNIX shell) because each unit behaves as an independent entity. When the client unit is integrated into a front-end desktop infrastructure that serves as an access point to the network-computing system (as with PUNCH), scalability can be achieved by supporting multiple front-ends. Single-point failures are eliminated by providing multiple access points via front-ends that present users with a consistent (i.e., identical) logical view of resources via shared or mirrored information.

At lower levels of the hierarchy, scalability with respect to users presents a logistical problem because the management and execution units will generally be distributed across different geographical regions and administrative domains. It is impractical to expect all users to have physical accounts on all the resources available to the network-computing sys-

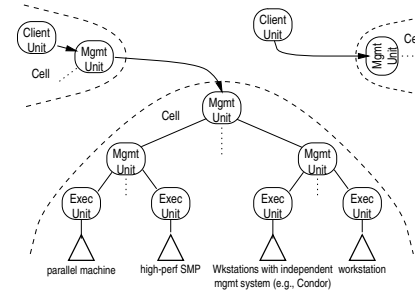


Figure 5. Management-unit hierarchy. Replication (not shown) eliminates single-point failures.

tem. This problem can be addressed by: 1) partitioning the system so that the individual components of the network-computing system are treated as “users” with respect to each other, and 2) viewing an “account” on the network-computing system as a logical entity (a capability) rather than a physical one. With logical accounts, the allocation and partitioning of physical resources (e.g., disk space) is done dynamically (on-demand). This allows the network-computing system to manage all of its users within a single physical account [31].³ The approach has the added advantage of allowing accounts to be clustered, while maintaining the separation of individual user data. For example, a PUNCH front-end only requires one logical account on any given management unit – even though the front-end itself maintains a separate logical account for each PUNCH user who has access to that front-end. Similarly, even though management units maintain separate logical accounts for each authorized front-end, each management unit only requires one logical account on any given execution unit. In practice, this ability to cluster logical accounts is extremely important because it gives administrators the freedom to add or delete local “users” without having to propagate the changes through the entire system.

Scalability with respect to software resources can be achieved by distributing them among an appropriate number of management units; support for replicated management units provides a degree of fault tolerance. Also, effective use of efficient data and information management strategies allows the number of tools per management unit to be maximized (see Section 3.3 for details).

Finally, scalability with respect to hardware resources and administrative domains can be achieved by allowing management units to forward requests to other management units (see Figure 5). Then, as the execution units increase, they can be distributed among multiple management units, and the management units can themselves be organized in a tree-structured hierarchy. (A request for tool execution originates at a client unit and goes through a sequence of management units until it is forwarded to an execution unit.) The hierarchy of management units also provides a mechanism that allows the network-computing system to

² These include specific protocols for tool invocation (e.g., standard sequential, PVM-based execution, etc.) and the like.

³ *Shadow* accounts (generic physical accounts that are rotated among users on demand) are required for some tools, depending on their characteristics.

be partitioned into independent *cells* (or sub-systems), as discussed in the next section.

2.4. Security and Access Control

Any infrastructure that spans multiple administrative domains must allow each domain to independently control its security and access control policies. Inter-domain security issues can be managed by partitioning the infrastructure into independent *cells*, each of which consists of at least one management unit and one execution unit (see Figure 5). Each cell is seen as a user by other cells. Execution units do not accept requests from management units in remote cells. Thus, a request from a remote cell must pass through a local management unit. A management unit that accepts a request originating from a different cell authenticates the immediate sender, and, optionally, the source of the request (the real user); this information can be propagated along with the request as a capability list, if so desired.

In a global environment, it is crucial to retain the ability to control the *visibility* of resources in addition to limiting access. This is true both from an administrative viewpoint and from a scalability perspective. In terms of administration, this ability is necessary because the networked environment spans multiple administrative domains, and organizations will generally not want internal resources to be visible to the outside world. In terms of scalability, the ability to customize the response according to the source of the request is beneficial because it relieves the overall system of the burden of managing sub-sets of resources that are unusable for a given class of users – this can result in a more efficient global “directory-of-services” function, for example. The ability to control the visibility of resources must be supported by customizing user-views directly at the associated management units – visibility cannot be limited in a secure manner by using filtering mechanisms on the client side.

2.5. Interface Issues

By definition, a network-computing system should be universally accessible. With the described architecture, this can be achieved by integrating the client unit component into the local computing environment (e.g., in a UNIX shell) or by providing a universally accessible front-end to the network-computing system. Given the current state of the art, universal access can be most easily achieved by allowing users to access the computing system via standard world-wide web browsers.

Designing a computing system interface that works within the framework of the existing web infrastructure and at the same time is flexible enough to accommodate the needs of diverse users and arbitrary tools presents some unique challenges (e.g., [6,13]). Issues that must be addressed include: 1) state management (WWW protocols are stateless), 2) support for existing tools without requiring access to source and/or object code, 3) support for interactive

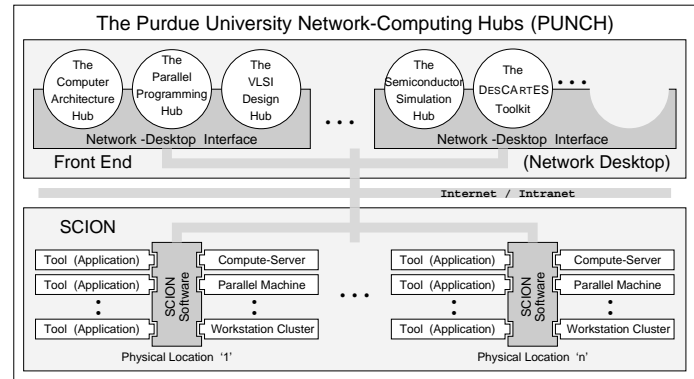


Figure 6. The PUNCH infrastructure.

text-based and graphical tool interfaces, and 4) input and output data (file) management. The computing system interface must also be transparent at the user level. This requires that the network-computing infrastructure appear as a single entity (i.e., provide a “single-system image”) in terms of commands issued and results displayed – which, in turn, implies support for (at least) location, replication, migration, and concurrency transparency. Finally, the interface must be *usable* [7] in the sense that it should have an intuitive “look and feel” and should provide adequate functionality. A detailed discussion of the solutions to these problems is beyond the scope of this paper; however, an overview of the approach used in PUNCH is presented in Section 3.5, and details are available in [31].

2.6. Performance Issues

The desirable performance advantages of a demand-based computing infrastructure over a conventional computing system are: 1) reduced job execution times, and 2) improved resource-utilization. Smaller execution times are a direct consequence of easy access to faster networked resources. Improved resource-utilization is a result of the demand-based infrastructure’s ability to customize its resource allocation policy to the anticipated resource-usage characteristics of a run. On the other hand, a network-computing system necessarily introduces additional overhead because of its distributed nature. The objective is to minimize this overhead by carefully designing the system architecture, and by using efficient protocols and data structures.

3. The PUNCH Infrastructure

The PUNCH infrastructure design closely follows the ideas presented in Section 2. PUNCH is made up of two parts: the front-end (network desktop) and SCION (see Figure 6). The front-end serves two primary functions: 1) it provides users with operating system services that can be accessed and used via standard world-wide web browsers, and 2) it user-transparently interfaces to one or more distributed computing systems that are capable of scheduling

and executing software applications across wide-area networks on demand. The client unit is an integral part of the front-end infrastructure. SCION serves as PUNCH's user-transparent middleware. It consists of a collection of hierarchically distributed servers (management units and execution units) that cooperate to provide the following services: 1) user-transparent management of the run-time environment, 2) independent control over resource access and visibility policies at each management unit node, and 3) cost and performance driven scheduling of available resources. As implied earlier, hardware resources managed by SCION can include different types of platforms, and software resources can consist of arbitrary tools.⁴ Resources can be located at any network-accessible site, and can be dynamically added or removed from the infrastructure.

3.1. PUNCH: A User's View

PUNCH is perceived by users as a computing system that provides universal, web-based access to tools. Functionally, it allows users to: 1) upload and manipulate input files, 2) run programs, and 3) view and download output – all via standard WWW browsers. For a detailed description of a user's view of PUNCH, see [29].

PUNCH can be logically divided into discipline-specific "hubs". Currently, PUNCH consists of four hubs that contain tools from semiconductor technology, VLSI design, computer architecture, and parallel processing. These hubs contain over thirty tools from eight universities and four vendors, and serve more than five hundred users mostly from Purdue, the US, and Europe.

Running a typical simulation on PUNCH is a three-step process. The first step involves the creation of the input file(s) required for the relevant simulation. In the second step, users define the input parameters (e.g., command-line arguments, etc.) for the program and start the simulation. Finally, after the simulation is complete, users can see, post-process, and download the results via the web-based front-end. PUNCH runs programs in a "background" mode by default. This means that the user's browser window is freed up as soon as the run has been successfully initiated.

3.2. System Architecture

The PUNCH architecture consists of network desktop interfaces, management units, and execution units (see Figure 7). The network desktop interfaces serve as access points into the network-computing system – they parse, interpret, and act on user-requests. This is the only level of the hierarchy that is visible to end-users. PUNCH's hierarchically distributed architecture is specifically designed to address the diverse demands imposed by the different types of requests processed by the system. Common types of transactions with relatively low demands are typically processed at the front-end. This allows the system to minimize the response

⁴ Tools with graphical user interfaces are supported with the help of display management technologies such as VNC [39].

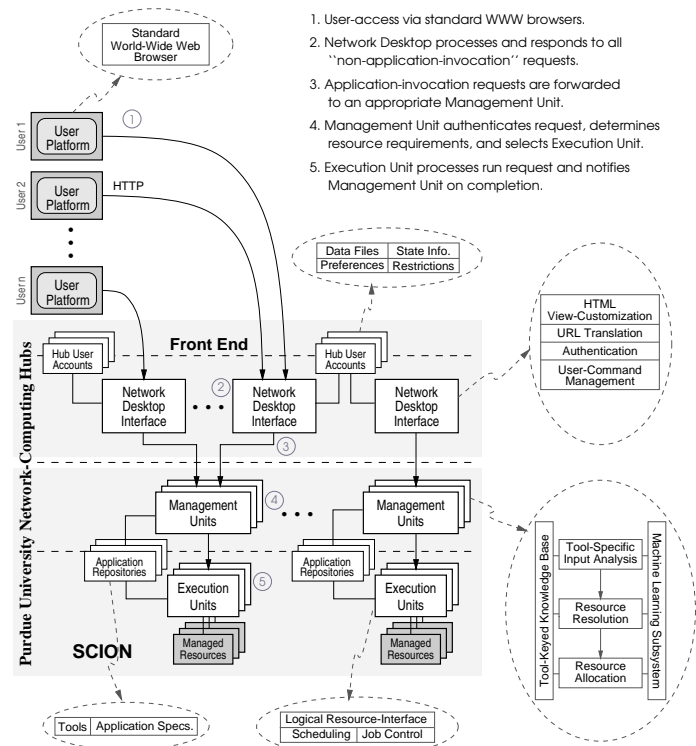


Figure 7. Architecture for PUNCH.

time for such transactions. Management units handle transactions that occur less frequently or have more complex requirements. In these situations, the higher latency caused by traversing multiple levels is either deemed acceptable or is effectively masked by the relatively long time required to process the (complex) transaction.

Figure 8 exemplifies the manner in which the PUNCH infrastructure could be distributed according to the needs of a given environment. Consider an organization 'O' that has a primary site 'M1' and satellite sites 'R1' - 'Rn'. The front-end (network desktop) will generally be established at each site because this helps minimize the network latency visible to PUNCH users and also allows each site to set its own account administration policies.

When a user is away from his/her site, he/she still has complete access to resources via a browser. In terms of security and access control, management units function as controlled gateways to the resources within a given administrative domain. Consequently, each organization will have at least one management unit. Finally, execution units provide management units with a parameterized interface to hardware resources or external resource management systems (e.g., Condor [34]). Thus, the network-computing system will have one execution unit for each managed resource.

The hierarchically distributed architecture facilitates centralized management of software and hardware resources, allowing the associated costs to be amortized across the entire organization. Moreover, it enables a business model in which infrequently-used resources are obtained from third-party sources – on a pay-per-use basis, for ex-

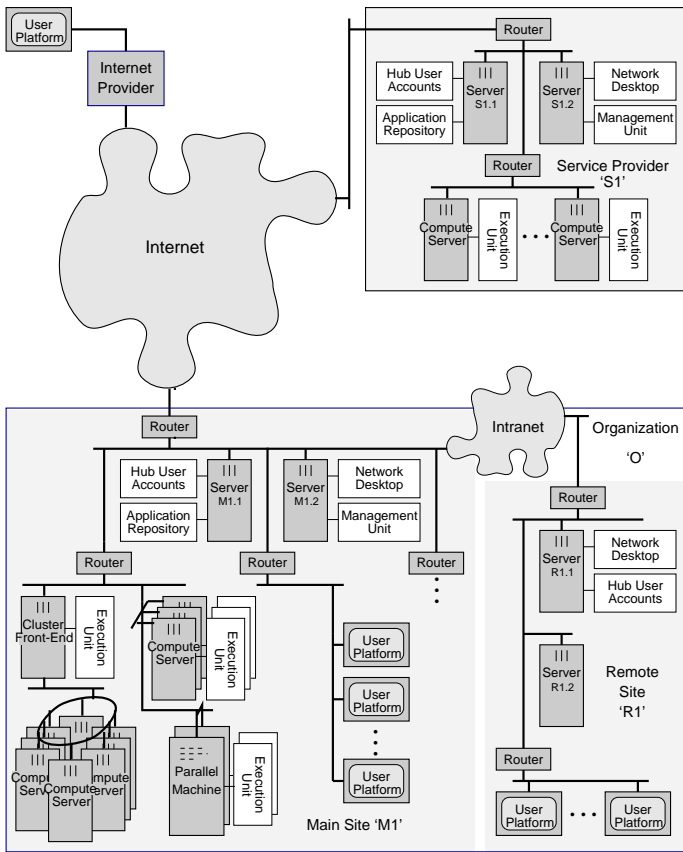


Figure 8. An example setup of PUNCH.

ample – in a user-transparent manner (see ‘S1’ in Figure 8).

PUNCH users have private and persistent data partitions on the front-end, much as UNIX users have private accounts. Users can create and edit files within their partition via PUNCH’s web-based interface. Once a file is in a user’s partition, it can be referenced by name via an appropriate field (e.g., a menu consisting of a list of files) in a HTML form interface.

3.3. Scalability and Reliability

Each component of the architecture can be independently replicated, based on the expected system load and the desired degree of redundancy. This contributes to the overall scalability and reliability of the network-computing system. At the component level, scalability issues are addressed by dynamically encoding meta-information into resource identifiers, as discussed below. The resulting *self-encoded* resource identifiers allow constant-time access to all meta-information managed by PUNCH. The results described in this section were obtained on a 300MHz Sun Ultra-4. In order to minimize network-related variations in the measured times, database files accessed during the experiments were stored on a local disk. PUNCH is implemented in Perl [45], and consequently runs in an interpreted environment. Thus, it is likely that the absolute values of the times re-

ported here are higher than those that could be obtained from a compiled version (implemented in ‘C’, for example).

In the course of processing any given transaction, PUNCH nodes access one or more databases in order to retrieve information associated with users and resources. Examples of such information include authentication and access control information for users, user-interface templates, learned resource usage characteristics, and portability information for hardware resources. It is extremely important to be able to access this information in a quick, efficient, and scalable manner. Information stored in database systems is retrieved by way of a query mechanism that typically involves a search. For large databases, the high latency associated with this search often translates to poor performance.

The PUNCH database system utilizes dynamic *access codes* to allow $O(1)$ access to the stored information. An access code represents the exact byte offset of a given record in a given database, and is dynamically encoded into all resource identifiers. For any query that includes an access code, the database system bypasses its normal search mechanism and performs a seek operation to a byte offset that is one less than the numeric value of the access code. It then reads the word at that position. If the supplied access code is correct, this word will be a record separator. Assuming that a record separator is found, the system reads the next word. PUNCH databases are organized so that the first word in every record consists of a string that uniquely identifies the record within that database. Typically, this string is the numeric value of the access code for that record; in a few cases, the name of the resource is used for the string when it (the name) is guaranteed to be unique (e.g., as with user logins). As a result of this organization, if the word read by the system matches the supplied access code (or the resource name when appropriate), the record being read is guaranteed to be the one that was requested. If the supplied access code is determined to be incorrect, the requested information is retrieved via a normal database search, and an access code is generated for use in subsequent queries.

Once obtained, access codes are cached at the source of the request, allowing them to be reused. The actual mechanism by which caching is achieved depends on the type of interface. For example, with the world-wide web interface, the access codes are cached implicitly by making them a part of the URL. On the other hand, when the communication is between PUNCH nodes, the access codes are cached explicitly on the client side.

Access codes can change over time – although this happens relatively infrequently in practice. (The change is generally a result of a resource being added to or removed from the database.) In order to minimize the search penalty when access codes are not available or are incorrect, resources are hashed across multiple databases on the basis of the initial characters of their identifiers and encrypted or encoded passwords. A schematic of the lookup mechanism used by the PUNCH database system is shown in Figure 9.

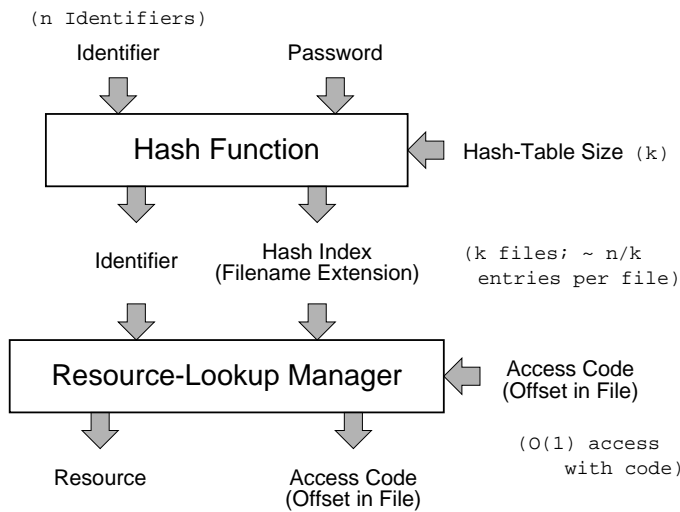


Figure 9. Schematic of the lookup mechanism used by the PUNCH database system.

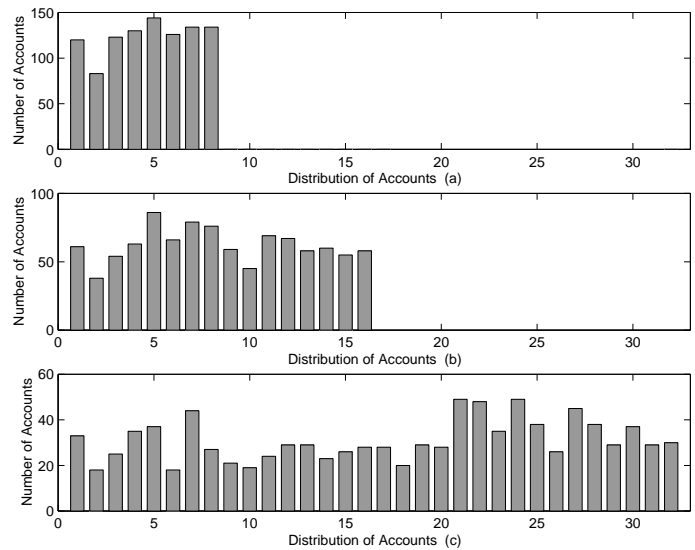


Figure 11. Distribution of user accounts achieved by the hash function for: a) eight, b) sixteen, and c) thirty-two buckets.

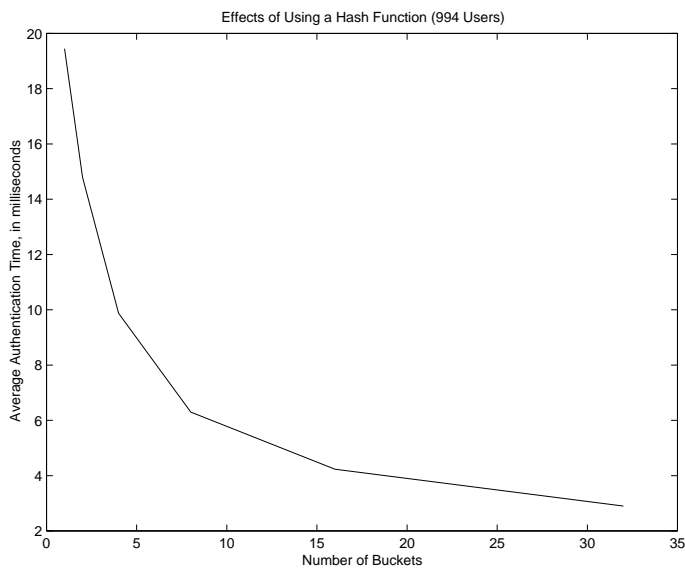


Figure 10. Effects of using a hash function on authentication time.

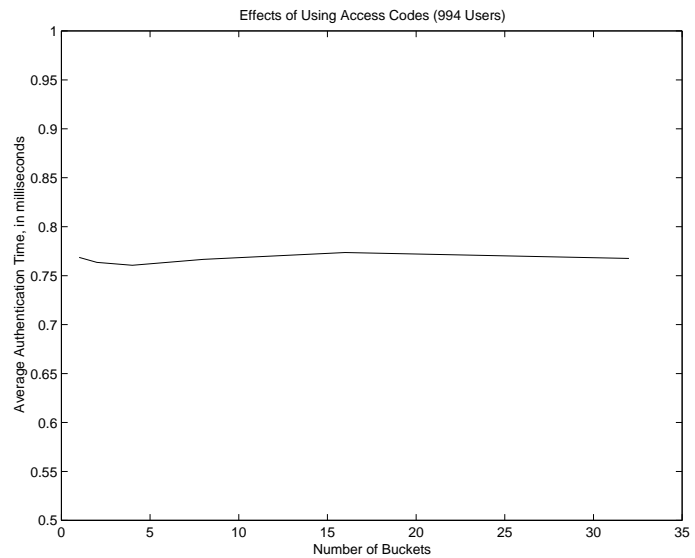


Figure 12. Effects of using access codes on authentication time.

The utility of the access codes is demonstrated by the following example. Authenticating a request from a PUNCH user involves looking up the appropriate password-file entry and verifying the password supplied with the request. A highly efficient authentication process is crucial to the scalability and performance of the overall system because the stateless nature of the PUNCH and HTTP communication protocols makes it necessary to individually authenticate each transaction. When access codes are not available or are incorrect, the authentication procedure involves a search. The latency associated with the search is minimized by distributing the user-account entries across multiple password files. The distribution is accomplished by way of a very simple hash function that selects a file on the basis of the ASCII values of the initial characters of the user's login and the encoded or encrypted password. Figure 10 illustrates the effects of the hash function on the average time required

to authenticate a request. The results are based on a *real* dataset consisting of 994 PUNCH user accounts. The plots in Figure 11 show that the (simple) hash function described above achieves an acceptably even distribution of accounts across available buckets for a real dataset.

Figure 12 shows the average time required to authenticate a request when correct access codes are available. The results represent the same dataset (994 PUNCH user accounts) used to generate the plot in Figure 10; compare the range of Y-axis values in the two plots. The utility of access codes can be illustrated more effectively by way of a synthetic dataset. Consider the information associated with tools (e.g., templates employed for user-interface generation, learned resource usage characteristics, portability information, etc.). In order to process any tool-related transaction, the run-time system must first locate the database

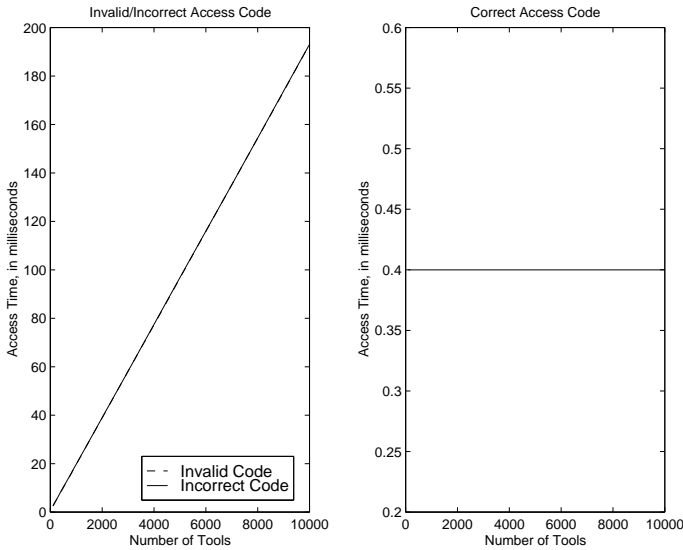


Figure 13. Effects of access codes on information retrieval latency.

Table 1
Observed access code proliferation.

Type of Use	Access Code	
	Present	Absent
Member Authentication	1,086,732	45,896
Internal References	946,652	81

for the tool. The scalability characteristics of the lookup time, when the access code is not known or incorrect, are shown in the left hand plot in Figure 13; the right hand plot shows the (constant) lookup time when a correct access code is supplied to the database system. Both plots were generated for a synthetic dataset using a database system with a linear search.

A typical request will contain several access codes – one for each database record that is accessed in the course of processing the corresponding transaction. For database records that consist of multiple sub-records, access codes are defined as a hierarchy of byte offsets. The byte offsets for sub-records are computed with respect to the parent record.

Table 1 shows the observed proliferation of access codes in the PUNCH environment. About 96% of the transactions initiated by member users contained access codes that could be used for authentication purposes. Access codes associated with tool-related information were present in virtually all transactions that involved such references, as indicated by the second row of the table.

3.4. Security and Access Control

PUNCH draws on the basic premise of virtual memory management to control access to its resources. Requests are assumed to contain virtual addresses (identifiers) that can be dynamically translated to physical ones. This model is used within the framework of the world-wide web by treating URLs as locations in a dynamic, virtual, and *active*

(side-effect based) address space. The client initiating a given transaction is equated to a distinct user-level process in a multi-user system, and the request is treated as an attempt to access a memory location with a lexicographical address within the private address space of the associated process.

Using this perspective, analogous to virtual memory management, each request can be verified for validity in terms of: 1) being within the address space bounds for the associated process (user), and 2) the type of access (read or write). *All* accesses to PUNCH undergo this validation process, allowing the system to control access to its resources on a per-user *and* a per-resource basis. Once the validity of a request has been verified, it is *mapped* to a physical resource. This mapping process is implicitly dependent upon the source (user-id) of the request. Thus, different users attempting to access the same (virtual) resource identifier (URL, in this case) could potentially trigger completely different actions.

PUNCH leverages this fact to dynamically generate a *logical* (virtual) view of available tools and resources for each class of users. For example, users who have member access to a given tool on PUNCH can view the corresponding on-line documentation by following an appropriate set of web links. Other users who follow these links are shown a different set of documents (typically a message stating that the user documentation for the tool is only available to authorized users). It is also possible to configure PUNCH so that such links are not generated at all for users who do not have member privileges for the corresponding tool.

3.5. Interface Issues

PUNCH supports tools with graphical or text-based interfaces. The tools do not have to be written in any particular language, and access to source or object code is not required. The PUNCH infrastructure can be distributed in a manner that allows tools to be (user-transparently) executed wherever they reside (currently, tools are executed at the following universities: Purdue, Illinois at Urbana-Champaign, Maryland, and Texas at Austin). Tools with graphical user interfaces are supported by leveraging display management technologies such as Broadway [9] and VNC [39]. (WinFrame [11] from Citrix could be leveraged in a similar manner.) Tools with text-based interfaces are supported by way of a programmable state machine that dynamically generates a customized HTML user interface for each program. The input to this state machine consists of: 1) a list of available *states*, and 2) a description of the *transitions* between these states (flow-control). Each state is specified in terms of an *HTML template*, which is essentially a standard HTML document with support for *variables* and *objects*. The flow-control information is specified via a high-level script language. The state machine can react to any run-specific information (e.g., values embedded within user-supplied strings, data within files, etc.), allowing

it to support programs that accept input in an interactive manner.⁵ Details are available in [31,32].

3.6. Performance Issues

Current computing systems typically execute a command by invoking a specific executable file that resides on a locally mounted filesystem. This mechanism implicitly assumes a one-to-one correspondence between user commands and local executable files. However, in practice, an executable file is merely one of (possibly) many different *implementations* of a given software application, where each implementation might be optimized for different subsets of input parameters (e.g., a parallel implementation may be best for large datasets) and system configurations, and might be compiled to run under particular operating systems on specific platforms. Also, most currently available computing systems attempt to execute tools on the machine on which the command was issued – without any regard to cost and performance criteria. PUNCH allows demand-driven management of existing software and hardware resources by delaying the binding of a user’s command to a specific implementation and machine until run-time, at which point the requirements of the given run can be analyzed.

For a computing system to be able to adapt to the needs of individual runs, it must have the ability to *estimate* the resource requirements of any given run *before* a scheduling decision is made. The resource usage characteristics of many tools depend on the run-time input to the tool. Also, in general, tools exhibit complex behavior that make analytical expressions describing the relationship between the run-time input and the corresponding resource usage characteristics nearly impossible. Consequently, PUNCH utilizes a *learning* approach to characterize this relationship (for each tool). Locally weighted polynomial regression (e.g., [4,12]) is used to predict the resource usage characteristics for each run; a detailed description of the machine learning system is available in [26,27].⁶ Finally, the prediction is used in conjunction with portability information to match the user’s request to the underlying network-accessible tools and resources. (The portability information consists of a list of the available implementations - e.g., sequential versus parallel - for a given tool, and the architectures on which they are supported.)

4. Conclusions

PUNCH is the result of a concerted effort to harness the existing networking and computing infrastructures and the rapidly-advancing world-wide web technologies with

the goal of building a *functional* demand-based network-computing infrastructure. Over the years, we have found the system to be an extremely useful resource for students and collaborators, and a highly flexible testbed for network-computing research. The ideas and solutions presented in this paper are based on (and validated by) our experiences in scaling PUNCH from a research project to a “live” system that is regularly used by several hundred students each semester.

The PUNCH infrastructure has been successfully implemented and applied to education, research, and technology-transfer. The earliest implementation of PUNCH was operational in April 1995. Since Fall 1996, PUNCH has been used to provide access to tools in several undergraduate and graduate courses, including a distance-education course at Purdue, a course each at the University of Berkeley and the University of Illinois at Chicago, and another at Technion in Israel. In addition, PUNCH was successfully leveraged to deliver a four-university technology-transfer short course (May 1997) for the Semiconductor Research Corporation (SRC). PUNCH serves as the underlying distributed computing infrastructure for two collaborative efforts involving five universities: the integration of design tools into new undergraduate and graduate curricula, and the Distributed Center for Advanced Electronics Simulations (DESCARTES). PUNCH is also the enabling infrastructure for a statewide Purdue University network-computing system currently being deployed.

Currently, PUNCH contains over thirty tools from eight universities and four vendors, and serves more than five hundred users from Purdue, across the US, and in Europe. During the past three years, PUNCH users have logged more than one million hits and have performed over seventy thousand simulations. Results from user-surveys indicate that the system performs well under the highly peaked usage patterns (very high usage in the hours before homeworks and projects are due) characteristic of an academic environment. (The current system configuration uses a dedicated HP-9000/C110 for the front-end, and distributes most runs among approximately ten shared compute-servers located at Purdue and other universities.)

5. Ongoing Work

Our current interests are directed at several issues central to network-computing research, including those of demand-based scheduling, resource monitoring, metacomputing and resource-aggregation, fault-management, and user-interface design. Work on scheduling is aimed at exploring the impact of a demand-driven environment (Section 3.6) on existing scheduling policies. Resource monitoring research is geared towards the design of a scalable system that can monitor and *predict* the availability and reliability characteristics of a large number of resources - the goal is to allow a demand-driven scheduler to request resources with specific characteristics (e.g., an Ultra-SPARC with 250MB memory that

⁵ Currently, only programs that have a static input tree (i.e., cases in which the queries generated by the program are only dependent on the user responses to preceding queries) are supported.

⁶ The current implementation predicts the CPU time and the network data-transfer time; memory and disk-space requirements will be predicted once the on-going development of a monitoring system is complete.

is likely to be free for the next 30 minutes). Metacomputing work is geared towards allowing users to logically “chain” distributed tools. Work on fault-management involves the ability to detect abnormal conditions and automatically take corrective actions. Finally, work on user-interface design is aimed at making network-computing transparent to end-users and tool installers.

Acknowledgements

This work was partially funded by the National Science Foundation under grants MIPS-9500673, CDA-9617372, EEC-9700762, ECS-9809520, and EIA-9872516. Special thanks are due to Mark S. Lundstrom for providing applications, leadership, support, and a user’s perspective for PUNCH.

References

- [1] C. Adasiewicz, Exploratorium: User friendly science and engineering, NCSA Access 9 (1995), 10-11.
- [2] P. Arbenz, W. Gander, and M. Oettli, The Remote Computation System, in: *High-Performance Computing and Networking (Lecture Notes in Computer Science, 1067)*, Springer-Verlag, 1996.
- [3] P. Arbenz, W. Gander, and M. Oettli, The Remote Computation System, *Parallel Computing* 23 (1997), 1421-1428.
- [4] C. G. Atkeson, S. A. Schaal, and A. W. Moore, Locally weighted learning, *AI Review* 11 (1997), 11-73.
- [5] J. E. Baldeschwieler, R. D. Blumofe, and E. A. Brewer, ATLAS: An infrastructure for global computing, in: *Proceedings of the 7th ACM SIGOPS European Workshop*, 1996.
- [6] S. Baker, V. Cahill, and P. Nixon, Bridging boundaries: CORBA in perspective, *IEEE Internet Computing* 1 (1997), 52-57.
- [7] R. G. Bias and D. J. Mayhew (eds.), *Cost-justifying usability*, Academic Press, 1994.
- [8] T. Brecht, H. Sandhu, M. Shan, and J. Talbot, ParaWeb: Towards world-wide supercomputing, in: *Proceedings of the 7th ACM SIGOPS European Workshop*, 1996.
- [9] X window system version 11 release 6.3: Release notes, Manual, X Consortium, Inc., 1996.
- [10] H. Casanova and J. Dongarra, NetSolve: A network solver for solving computational science problems, in: *Proceedings of the Supercomputing Conference*, 1996.
- [11] Citrix Systems, Inc., ICA technical paper, WWW document at www.citrix.com/technology/, 1996.
- [12] W. S. Cleveland and S. J. Devlin, Locally weighted regression: An approach to regression analysis by local fitting, *Journal of the American Statistical Association* 83 (1988), 596-610.
- [13] S. E. Dossick and G. E. Kaiser, WWW access to legacy client/server applications, Technical report CUCS-003-96, Columbia University (1996).
- [14] S. E. Dossick, G. E. Kaiser, and J. J. Yang, Distributed tool services via the world wide web, Technical report CUCS-042-96, Department of Computer Science, Columbia University (1996).
- [15] I. Foster and C. Kesselman, Globus: A metacomputing infrastructure toolkit, *International Journal of Supercomputer Applications* 11 (1997).
- [16] I. Foster and C. Kesselman, The Globus project: A status report, in: *Proceedings of the 1998 Heterogeneous Computing Workshop (HCW'98)*, 1998, pp. 4-18.
- [17] A. S. Grimshaw, A. Nguyen-Tuong, and W. A. Wulf, Campus-wide computing: Early results using Legion at the university of virginia, Technical report CS-95-19, Department of Computer Science, University of Virginia (1995).
- [18] A. S. Grimshaw and W. A. Wulf, Legion: Flexible support for wide-area computing, in: *Proceedings of the 7th ACM SIGOPS European Workshop*, 1996.
- [19] A. S. Grimshaw, W. A. Wulf, and et al., The Legion vision of a worldwide virtual computer, *Communications of the ACM* 40 (1997).
- [20] O. Gunther, R. Muller, P. Schmidt, H. K. Bhargava, and R. Krishnan, MMM: A web-based system for sharing statistical computing modules, *IEEE Internet Computing* 1 (1997), 59-68.
- [21] P. Homburg, M. v. Steen, and A. S. Tanenbaum, An architecture for a wide area distributed system, in: *Proceedings of the 7th ACM SIGOPS European Workshop*, 1996.
- [22] P. T. Homer and R. D. Schlichting, Constructing scientific meta-computations, in: *Proceedings of the International Conference on High-Performance Computing in the Asia-Pacific Region (HPC'Asia)*, 1995.
- [23] P. T. Homer and R. D. Schlichting, Configuring scientific applications in a heterogeneous distributed system, *Distributed Systems Engineering* 3 (1996), 173-184.
- [24] D. Israel, NCSA Symbio: A supercomputer for windows NT, in: *NCSA Access Online*, NCSA, 1997.
- [25] G. E. Kaiser, S. E. Dossick, W. Jiang, J. J. Yang, and S. X. Ye, WWW-based collaboration environments with distributed tool services, *World Wide Web Journal* (1998).
- [26] N. H. Kapadia, C. E. Brodley, J. A. B. Fortes, and M. S. Lundstrom, Resource usage prediction for demand-based network-computing, Technical report TR-ECE 98-9, Department of Electrical and Computer Engineering, Purdue University (1998).
- [27] N. H. Kapadia, C. E. Brodley, J. A. B. Fortes, and M. S. Lundstrom, Resource-usage prediction for demand-based network-computing, in: *Proceedings of the 1998 Workshop on Advances in Parallel and Distributed Systems (APADS)*, 1998.
- [28] N. Kapadia, J. A. B. Fortes, and M. Lundstrom, The Computational Electronics Hub: A network-based simulation laboratory, in: *Summary Record of the Workshop on Materials and Process Research and the Information Highway*, 1996, pp. 31.
- [29] N. H. Kapadia, J. A. B. Fortes, and M. S. Lundstrom, The Semiconductor Simulation Hub: A network-based microelectronics simulation laboratory, in: *Proceedings of the 12th Biennial University Government Industry Microelectronics Symposium*, 1997, pp. 72-77.
- [30] N. H. Kapadia and J. A. B. Fortes, On the design of a demand-based network-computing system: The Purdue University Network-Computing Hubs, in: *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC'98)*, 1998, pp. 71-80.
- [31] N. H. Kapadia and J. A. B. Fortes, The PUNCH network desktop, Technical report TR-ECE 99-1, Department of Electrical and Computer Engineering, Purdue University (1999).
- [32] N. H. Kapadia, J. P. Robertson, and J. A. B. Fortes, Interface issues in running computer architecture tools via the world-wide web, in: *Proceedings of the Workshop on Computer Architecture Education at the 25th Annual International Symposium on Computer Architecture (ISCA'98)*, 1998.
- [33] S. B. Lamine, J. Plaice, and P. Kropf, Problems of computing on the web, in: *Proceedings of the 1997 High Performance Computing Symposium*, 1997, pp. 296-301.
- [34] M. Litzkow, M. Livny, and M. W. Mutka, Condor a hunter of idle workstations, in: *Proceedings of the 8th International Conference on Distributed Computing Systems*, 1988, pp. 104-111.
- [35] J. Martin, *Computer networks and distributed processing: Software, techniques, and architecture*, Prentice-Hall, 1981.
- [36] F. Ramme, Building a virtual machine-room a focal point in metacomputing, *Future Generation Computer Systems* 11 (1995), 477-489.
- [37] A. Reinefeld, R. Baraglia, T. Decker, J. Gehring, D. Laforenza,

- F. Ramme, T. Romke, and J. Simon, The MOL project: An open, extensible metacomputer, in: *Proceedings of the 1997 IEEE Heterogeneous Computing Workshop (HCW97)*, 1997, pp. 17-31.
- [38] F. D. Reynolds, Evolving an operating system for the web, *Computer* 29 (1996).
- [39] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, Virtual network computing, *IEEE Internet Computing* 2 (1998), 33-38.
- [40] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi, Nin: A network based information library for global world-wide computing infrastructure, in: *High-Performance Computing and Networking (Lecture Notes in Computer Science, 1225)*, Springer-Verlag, 1997, pp. 491-502.
- [41] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima, -Nin-: Network-based information library for globally high performance computing, in: *Proceedings of Parallel Object-Oriented Methods and Applications (POOMA)*, 1996.
- [42] D. Souder, M. Herrington, R. P. Garg, and D. DeRyke, JSPICE: A component-based distributed Java front-end for SPICE, in: *Proceedings of the 1998 Workshop on Java for High-Performance Network Computing*, 1998.
- [43] M. v. Steen, P. Homburg, and A. S. Tanenbaum, The architectural design of Globe: A wide-area distributed system, Technical report IR-422, Department of Mathematics and Computer Science, Vrije Universiteit (1997).
- [44] V. Sunderam, Heterogeneous network computing: The next generation, *Parallel Computing* 23 (1997), 121-135.
- [45] L. Wall, T. Christiansen, and R. L. Schwartz, *Programming perl*, O'Reilly & Associates, 1996.



Nirav H. Kapadia received his B.E. degree in Electronics and Telecommunications from Maharashtra Institute of Technology (India) in 1990. He subsequently worked at the institution for a year as a lecturer. He received his M.S.E.E. degree from Purdue University in 1994, and is currently finishing up his doctoral dissertation at Purdue University under the supervision of professors Jose Fortes and Mark Lundstrom. He has started working as a Senior Research Scientist in

the School of Electrical and Computer Engineering at Purdue University as of January 1999.

Kapadia was the primary architect of the Purdue University Network Computing Hubs (PUNCH), a distributed network-computer that provides geographically dispersed users with universal, web-based access to unmodified tools. Kapadia's research interests include network-computing system design, demand-based resource management in a wide-area distributed computing environment, metacomputing, machine learning techniques for resource usage prediction, and user-interface management in a networked environment. He has coauthored ten technical papers.

E-mail: kapadia@purdue.edu



José A. B. Fortes received the B.S. degree in Electrical Engineering (Licenciatura em Engenharia Electrotécnica) from the Universidade de Angola in 1978, the M.S. degree in Electrical Engineering from the Colorado State University, Fort Collins in 1981 and the Ph.D. degree in Electrical Engineering from the University of Southern California, Los Angeles in 1984. In 1984, he joined the faculty of the School of Electrical Engineering at Purdue University, West Lafayette, Indiana,

where he currently is a Professor. From July 1989 through July 1990 he served at the National Science Foundation as director of the Microelectronics Systems Architecture program. From June 1993 till January 1994 he was a Visiting Professor at the Computer Architecture Department of the Universitat Politècnica de Catalunya in Barcelona, Spain.

His research interests are in the areas of parallel processing, computer architecture, network-computing and fault-tolerant computing. He has authored or coauthored over 100 technical papers. Fortes is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) professional society.

E-mail: fortes@purdue.edu