Active Yellow Pages: A Pipelined Resource Management Architecture for Wide-Area Network Computing

Dolors Royo[†] dolors@ac.upc.es
Nirav H. Kapadia[‡] kapadia@purdue.edu
José A. B. Fortes[‡*] fortes@purdue.edu
Luis Diaz de Cerio[†] ldiaz@ac.upc.es

[†]Departemento de Arquitectura de Computedores
Universitat Politècnica de Catalunya, Barcelona, Spain

[‡]School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285, USA

Abstract

This paper describes a novel, pipelined resource management architecture for computational grids. The design is based on two key realizations. One is that resource management involves a sequence of tasks that is best handled by a pipeline. As shown in the paper, this approach results in a scalable architecture for decentralized scheduling. The other realization is that static aggregation of resources for improved scheduling is inadequate in wide-area computing environments because the needs of users and jobs change with both, location and time. The described architecture addresses this problem by dynamically aggregating resources in a manner that continuously optimizes system response. This is accomplished by way of an active yellow pages directory that allows aggregation constraints to be (re)defined on the fly. An initial prototype of the active yellow pages service has been deployed in the PUNCH network computing environment. Experiences with the production PUNCH system and preliminary results from controlled experiments indicate that the active yellow pages service performs well.

1. Introduction

Network-centric computing promises to revolutionize the way in which computing services are delivered to the end-user. Analogous to the power grids that distribute electricity today, *computational grids* will distribute and deliver computing services to users anytime, anywhere. At the heart of the computational grid is an ability to harness, manage, and channel distributed compute cycles, software, and data according to demand.

[*]At the Department of ECE, University of Florida from September 2001.

Resource management systems designed for computational grids must support three key capabilities: 1) they must provide support for decentralized scheduling decisions and distributed access control, 2) they must be able to interoperate with local scheduling subsystems, and 3) they must be *self-optimizing* in the sense that they must be able to dynamically adapt to changing workloads and resource usage constraints.

The first capability is necessary in order to allow sites to retain control over their local resources even when they are a part of a wide-area computational grid. The second capability is crucial from a practical standpoint — it allows site-specific solutions to be quickly integrated into a computational grid. Finally, the third capability is necessary because — in wide-area computing environments — user-requirements, application-demands, and available resources tend to change with both, location and time, making it difficult to manually "tweak" the system to improve performance.

This paper describes a novel, *pipelined resource management architecture* that is designed for use in computational grids that span multiple administrative domains. The architecture has three key features. First, it is designed to dynamically adapt to the requirements of the observed mix of jobs — this is accomplished by way of an *active yellow pages* directory that allows resources to be dynamically aggregated in a manner that continuously optimizes system response. Second, the pipelined architecture results in a scalable and flexible resource management system with built-in support for redundancy — this is achieved by allowing individual stages of the pipeline to be independently

0-7695-1296-8/01 $10.00 © 2001 IEEE

distributed and replicated. Finally, the architecture lends itself to decentralized control and a "system of systems" approach to resource management — each stage in the pipeline treats the preceding stage as a user that is subject to authentication and policy constraints.

The emphasis of the work so far has been on designing a decentralized resource management architecture for systems such as the Purdue University Network Computing Hubs (PUNCH) [17, 15]. An initial prototype of the active yellow pages service has been deployed on the production PUNCH system and preliminary results indicate that it works well. However, further evaluation is necessary — and is the subject of ongoing work.

The paper is organized as follows. Section 2 outlines the role played by the active yellow pages service in the PUNCH network computing environment. Section 3 describes the architecture of PUNCH from a resource management viewpoint. Section 4 outlines the different sub-systems that make up the active yellow pages service. Section 5 describes the resource management pipeline and the associated query language. Section 6 provides a qualitative discussion of the key benefits of using a pipelined architecture and active yellow pages for resource management in a computational grid. Section 7 presents preliminary results for a prototype implementation of the architecture. Section 8 places the described research in context with related work. Finally, Section 9 presents the conclusions of this work and outlines future directions.

2. The PUNCH Network Computer

Delivering computing as a service requires that the underlying infrastructure be able to negotiate resources between institutional boundaries — much as electricity is bartered among different utility companies. For example, consider a user who wants to run an application from a given vendor on data that happens to reside at a remote storage warehouse. In the PUNCH environment, the user connects to a *network desktop* via a standard Web browser, provides the "location" of his/her storage service provider, and clicks on the application of interest.[1] At this point, the network desktop must identify and locate appropriate resources, and *assemble* the necessary computing environment for the user [16].

This task is accomplished as follows. The network desktop first verifies that the user is authorized to run the selected application. Next, it uses the active yellow pages (ActYP) service described in this paper to identify, locate, and select appropriate compute server(s) for the run. The ActYP service also selects available *shadow accounts* [16] in which to run the application; shadow accounts are not explicitly tied to any individual user. Then, the virtual file system service [7] mounts the application and data disks on to the selected machine. Finally, the application is invoked on the selected machine and, for applications with graphical user-interfaces, the display is routed to the user's browser via remote display management technologies such as VNC [23]. Once the run is complete, the virtual file system service unmounts the application and data disks, and the network desktop relinquishes the shadow account and resources by notifying the ActYP service.

The key value of the active yellow pages service in such environments is its ability to 1) support decentralized resource management decisions and access control policies, and 2) hide site-specific configurations and policies from the core network computing infrastructure. The network desktop simply asks ActYP for resources (via a query language); and it gets back an IP address, a TCP port number, and a session-specific access key. ActYP negotiates for the resources, verifies that relevant services are available and starts daemons as necessary, allocates shadow account `uids` on compute servers as appropriate, and facilitates the exchange of session-specific authentication information among resources that are dispersed across different administrative domains.

A prototype of the active yellow pages service described in this paper has been in use for about one year. PUNCH currently has about 2,000 users across two dozen countries, and offers access to more than 70 engineering applications. PUNCH can be accessed at `www.punch.purdue.edu`.

3. PUNCH System Architecture

From a resource management perspective, PUNCH can be divided into three main components: the network desktop, the application management component, and the active yellow pages service (see Figure 1). With reference to the figure, users interact with PUNCH via its Web-accessible network desktop (event 1 in the figure). The network desktop processes file- and data-manipulation requests locally, and forwards requests for tool execution to an application management component (event 2 in the figure). As shown in Figure 2, the application management component parses the user input, extracts relevant parameters based on information in a knowledge base, estimates the run-

[1] Currently, the storage location is implicitly configured when a user requests a PUNCH account.

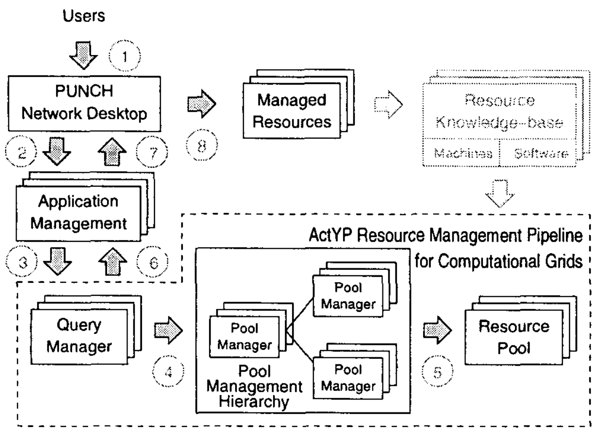

Figure 1. The components of the PUNCH infrastructure from a resource management perspective. The numbers 1 - 6 in the figure show the sequence of events that occur in the process of scheduling and initiating a run on PUNCH. Details are provided in the text.

time for the application (via a performance modeling service; see [14, 18] for details), determines software and hardware requirements, and constructs a query for the active yellow pages (ActYP) service from the available data. The generated query is subsequently forwarded to the ActYP service (event 3 in Figure 1).

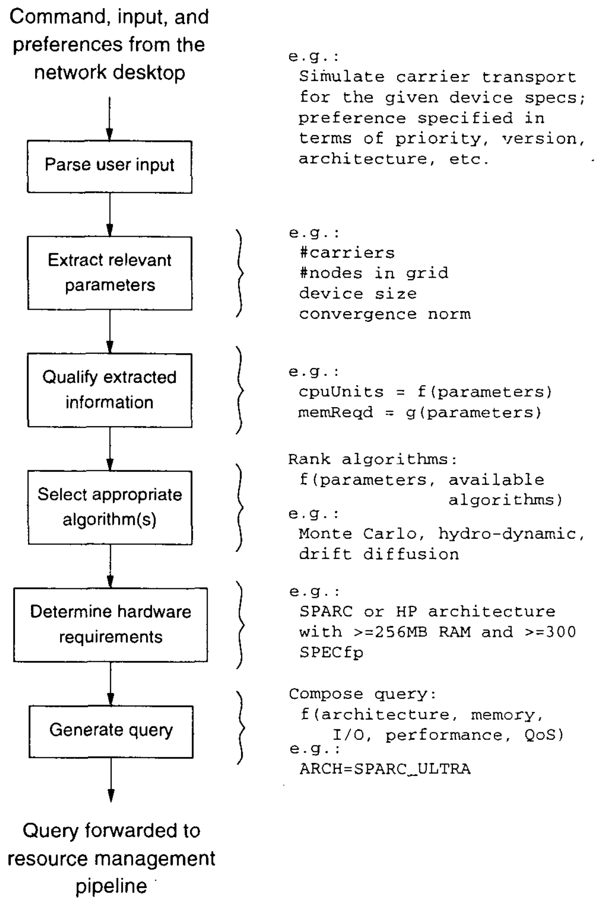
Command, input, and preferences from the network desktop
Parse user input
e.g.: Simulate carrier transport for the given device specs; preference specified in terms of priority, version, architecture, etc.
Extract relevant parameters
e.g.: #carriers #nodes in grid device size convergence norm
Qualify extracted information
e.g.: cpuUnits = f(parameters) memReqd = g(parameters)
Select appropriate algorithm(s)
Rank algorithms: f(parameters, available algorithms) e.g.: Monte Carlo, hydro-dynamic, drift diffusion
Determine hardware requirements
e.g.: SPARC or HP architecture with >=256MB RAM and >=300 SPECfp
Generate query
Compose query: f(architecture, memory, I/O, performance, QoS) e.g.: ARCH=SPARC_ULTRA
Query forwarded to resource management pipeline

Figure 2. An overview of the scheduling events that occur within the application management component shown in Figure 1.

4. The Active Yellow Pages Service

Resource management in heterogeneous computing environments involves three key tasks: 1) identifying the types of resources appropriate for a given run, 2) *locating* those types of resources, and 3) selecting appropriate instances of the located resources.

The first task is performed by the entity requesting the resources — the application management component in the case of PUNCH, as outlined in the previous section. The second task involves a search that is often accomplished by way of a directory service (e.g., Globus employs the Metacomputing Directory Service [8]). The third task involves the use of appropriate scheduling algorithms (e.g., [1, 13, 24]) to select the "best" of the available resources.

The second task outlined above is typically accomplished by going through a "database". This search is analogous to going through the "white pages" listings of a telephone directory. The task of locating *specific types of resources*, however, is more suited to a "yellow pages" lookup, where listings are grouped according to some criteria. This leads to the basic idea of establishing a yellow pages service for resource management.

Traditional yellow pages directories are based on the implicit assumption that the listings can be classified according to fixed and well-established criteria (e.g., airlines, hotels, etc.). In a computing environment, however, it is impractical to anticipate all possible permutations for the characteristics that define a resource. This leads to the notion of an *active* yellow pages directory, where the categories are defined on the fly.

The PUNCH active yellow pages (ActYP) service is made up of three cooperating sub-systems: 1) one or more directory services or resource databases that maintain information about resources in the computational grid, 2) a resource monitoring service that keeps track of the state of the resources, and 3) a re-

```
1.    resource state
2.    current load
3.    active jobs
4.    available memory
5.    available swap
6.    time of last update
7.    PUNCH service status flags
8.    effective speed
9.    number of CPUs
10.   maximum allowed load
11.   machine name
12.   machine object pointer
      (access and audit information)
13.   shared account identifier
14.   execution unit port
15.   PVFS mount manager port
      (see [16] for details)
16.   user group list
      (list of allowed user groups)
17.   tool group list
      (types of tools supported by machine)
18.   shadow account pool pointer
      (see [16] for details)
19.   usage policy pointer
20.   administrator defined parameter list
```

Figure 3. A list of the fields maintained by the PUNCH resource database for each machine.

source management pipeline that dynamically aggregates "similar" resources in a manner that optimizes scheduling response times. This section describes the first two sub-systems in the context of ActYP; the third sub-system forms the heart of ActYP, and is described in the rest of the paper.

4.1. Directory services

PUNCH currently uses a custom database that accommodates the needs of the operational portal and, at the same time, facilitates the evaluation of the active yellow pages service.[2] For each resource (i.e., machine), the database maintains several fields, as shown in Figure 3.
The first field represents the state of the system, and can have one of three values: `up`, `down`, or `blocked`. Fields 2 - 7 contain information required by the PUNCH scheduler, and are dynamically updated by a resource monitoring system. Fields 8 - 11 contain relatively static information about the machine; these fields are currently updated manually.

[2]A description of the design of the database is beyond the scope of this paper.

The machine object pointer (field 12) is a path to a file that contains access and audit information for the machine (e.g., ssh key, owner information, instructions for starting a PUNCH server on the machine, etc.). The shared account identifier lists the name of a shared account on the machine (e.g., user *nobody*), if any.[3]

The execution unit port identifies the TCP port at which the PUNCH execution unit (see [17] for details) is running in the shared account (if it exists) on the corresponding machine. The PVFS mount manager port (field 15) lists the TCP port at which the mount manager of the PUNCH Virtual File System service [16] can be contacted.

The user group list (field 16) identifies the types of users who are allowed to use the corresponding machine, and the tool group list (field 17) enumerates the types of tools that the machine is able to run. The shadow account pool pointer references a secondary database that manages shadow accounts [16] available to PUNCH on that machine. The usage policy field is currently unimplemented, but it is designed to point to a PUNCH metaprogram [19] that would allow administrators to specify complex usage policies (e.g., public users are only allowed to access this machine if its load is below a specified threshold).

Finally, field 20 allows administrators to specify arbitrary key-value pairs that are used by the active yellow pages service at run-time as described in the next section. Parameters typically used in the current PUNCH system include the following: arch (architecture), memory, ostype, osversion, owner, swap, and cms (supported cluster management systems; e.g., `cms=sge,pbs,condor`).

4.2. Resource monitoring

The primary function of the resource monitoring system is to update fields 2 - 7 in the database. Almost any available resource monitoring system can be used to provide the necessary functionality.[4]

5. Resource Management Pipeline

5.1. Query language

Queries received by the resource management pipeline describe the following: resource requirements,

[3]This account, if it exists, is used by PUNCH to run applications/utilities identified as "safe" by local system administrators. The primary benefit of using a shared account is to improve the response time for very short jobs.

[4]An open source version of the performance co-pilot from SGI (www.sgi.com/software/co-pilot/) is currently being evaluated in the context of PUNCH.

predicted application behavior, and user-specific data. Resource requirements include, for example, system architecture, operating system type and version, minimum memory, and software license constraints. Information about application behavior, when available, consists of estimates of the resources (e.g., CPU time and memory usage) that will be needed for the particular run [14, 18]. User-specific information includes parameters such as login, access group, and access keys or passwords. The following is an example of a relatively simple query generated by PUNCH:

```
punch.rsrc.arch = sun
punch.rsrc.memory = >=10
punch.rsrc.license = tsuprem4
punch.rsrc.domain = purdue
punch.appl.expectedcpuuse = 1000
punch.user.login = kapadia
punch.user.accessgroup = ece
```

The query requests a "sun" machine with at least ten megabytes (the default unit) of memory and a license for an application that is identified as "tsuprem4''. It further specifies that the machine must be within the "purdue" domain. The query also states that the run is expected to take one thousand CPU seconds[5] and contains the login and access group of the user attempting to initiate the particular run.

The query language used by the resource management pipeline employs a hierarchical namespace for the keys in the key-value pairs. In the example above, the family "punch" defines the semantics for the types "rsrc", "appl", and "user". Valid words for the final part of the key and the interpretation of the value part of the key-value pairs (e.g., numeric, string, range, etc.) are specified by administrators as described in the previous section. For queries in the punch family, when a key of type rsrc (for example, punch.rsrc.ostype) is not specified, its value defaults to "don't care". For missing keys of type appl and user, the values default to "undefined". New families of key-value pairs could be defined to allow the resource management pipeline to simultaneously support multiple protocols and semantics: this could allow ActYP to reuse Condor's ClassAds [22], for example.[6]

[5]The current protocol assumes the existence of a "reference" machine for time-related estimates. In the future, the protocol will be extended to include relevant meta-information — for example, one could specify the expected CPU time as "1000s@sun4u:sparc:ultra-5_10:333MHz" and include multiple estimates when appropriate.

[6]Only the punch family is implemented currently.

5.2. Pipeline architecture

This section describes the different stages of the ActYP resource management pipeline architecture (see Figure 1). In brief, *query managers* receive queries from clients (event 3 in the figure), decompose them into basic components, and forward them to appropriate *pool managers* (event 4 in the figure). Pool managers map queries to pool names and forward the queries to appropriate *resource pools* (event 5 in the figure). They also create resource pools when necessary. Resource pools are active objects that consist of 1) machines aggregated according to a specified criteria (e.g., architecture, memory, and/or owner) and 2) processes or threads that order the machines on the basis of a specified scheduling objectives. On receiving a query, resource pools allocate appropriate machine(s) and forward the information to the requesting client (event 6 in the figure). The client then initiates the application on the selected machine(s) (event 8 in the figure).

5.2.1. Query managers

Queries enter the resource management pipeline via a query manager stage (event 3 in Figure 1). Query managers translate queries into a standard internal format, decompose *composite queries* into basic components, select appropriate pool managers, and forward queries to the selected pool managers. Each of these steps is described below.

Query translation. Translating queries into a predefined internal format is an effective way of supporting interoperability. This allows different network-computing systems to query the pipeline using their native resource specification languages as long as an appropriate translator has been implemented in the query manager. The key-value-based query language described in the previous section serves as the native language for the resource management pipeline.

Composite queries. A composite query is one which contains "or" clauses. Such queries are decomposed into multiple basic queries that are processed concurrently by subsequent stages of the pipeline. The process of decomposing queries at the beginning of the pipeline and reintegrating the results at the end is analogous to the fragmentation of datagrams in TCP/IP [5]; appropriate state information is propagated along with each query component in order to allow reintegration at the end of the pipeline. For example, a query that requests a machine with either a "sun" or an "hp" architecture will be decomposed into two basic queries — one for a sun machine and one for an hp. The two queries will be simultaneously forwarded to (possibly different) pool managers. At the end of the pipeline, the results generated by the basic queries will be reintegrated within another query manager stage (not shown in Figure 1) and returned to the client.

Pool manager selection. Query managers select pool managers on the basis of the values of one or more of the parameters specified within queries. It is also possible to select pool managers in random or round-robin order. As an example, a query manager can be configured to select one set of pool managers for `sun` machines and a different set for `hp` machines; an individual pool manager from a particular set can be selected randomly.

5.2.2. Pool managers

Pool managers map queries to pool names and select an appropriate instance of a resource pool when multiple ones exist. They also create resource pools when necessary, and forward queries to other pool managers if the requested resources are not available locally. Each of these steps is described below.

Mapping queries. A pool name is made up of two components: a *signature* and an *identifier*. Thus, the mapping process requires pool managers to construct a signature and an identifier for each query. The signature is constructed by forming a colon-separated list of sorted `rsrc` keys in the query, and a string that specifies the corresponding comparative operators (e.g., equal to, greater than, etc.). The identifier is constructed by forming a colon-separated list of the values associated with the sorted `rsrc` keys that make up the signature. Thus, for the sample query in Section 5.1, the signature is `arch:domain:license:memory,==:==:==:>=` and the identifier is `sun:purdue:tsuprem4:10`. The second part of the signature represents the "equal-to" and "greater-than-equal-to" operators in the query.

Resource pool selection. Pool managers keep track of resource pools via a local directory service. Once a query has been mapped to a pool name, the pool manager uses the directory service to retrieve pointers (i.e., machine names and TCP/UDP ports) to all instances of resource pools with the particular name. It then randomly selects one of the instances and forwards the query to that resource pool.

Resource pool management. If an instance for a resource pool with a particular name does not exist, pool managers attempt to create a new instance (the actual process of creating a resource pool is described in the next section). If one cannot be created, the pool manager attaches its own name to a list within the query, decrements a "time-to-live" counter associated with the query, and forwards it to one of the pool managers listed in the local directory service. The list of names attached to the query prevents it from being sent to any given pool manager more than once. The time-to-live counter is analogous to the TTL field in IP packets [5]; the request is considered to have failed when the counter reaches zero.

5.2.3. Resource pools

Resource pools are dynamically-created "objects" that consist of 1) machines aggregated according to specified criteria (e.g., software, user group, machine architecture, etc.), and 2) processes (or threads) that order the machines on the basis of specified scheduling objectives. The following discussion explains the mechanisms used to create and initialize these objects, and how machines within these objects are scheduled.

Creating new resource pools. Pool managers create new resource pools. If the resource pool and the pool manager are on the same machine, the pool manager simply forks a process that initializes itself and listens to a specified port. If the resource pool is on a different machine, the pool manager starts it via a proxy server on the remote machine. (This server is a part of the ActYP service, and is assumed to be kept alive via a `cron` process.)

Initializing pool objects. The pool object first walks the "white pages" database for machines that match the criteria encoded within its name. During this process, the pool object loads relevant information (machine name, in the current implementation) about appropriate machines into a local cache and marks them as "taken" within the main database. Once initialization is complete, the pool object makes itself available to pool managers by registering its name and a self-generated instance-number with the local directory service.

Scheduling mechanisms. Each pool object has one or more scheduling processes associated with it. The function of these processes is to sort machines within the object's cache using specified criteria (e.g., average load or available memory), and to process queries sent by pool managers. Pool objects can be configured to utilize different scheduling objectives [20] and policies.

6. Qualitative Analysis

The previous sections described the active yellow pages service and its pipelined architecture. This section outlines the key benefits of this architecture in terms of metrics that are relevant in computational grid environments.

Scalability, reliability, and redundancy. All stages in the resource management pipeline can be independently distributed and replicated across machines. Queries propagate from one stage to the next via TCP or UDP. Within a given administrative domain, replicated instances share information via directory services and databases. A key benefit of the pipelined architecture is that stages that become bottlenecks can be replicated — thus allowing "hot spots" to be addressed without needing to reconfigure the entire system. The pipeline also provides a degree of decoupling between different types of queries.

Support for QoS negotiation. The pipelined resource management architecture provides inherent support for multiple levels of quality of service. For example, higher levels of QoS could be provided by simultaneously forwarding a given query to multiple pool managers and pool objects, and utilizing the best response. In contrast, the response time for composite queries could be minimized by returning the first available match — as opposed to waiting for results from different components to be reintegrated. Improved quality of service can also be achieved by using better or more sophisticated heuristics to select instances of pool managers and pool objects.

Self-optimizing resource management. Large computing environments often exhibit a temporal locality of runs. This is particularly true of academic settings — students working on assignments will all use certain applications over and over within a relatively short period of time. The described architecture exploits this locality by dynamically aggregating resources on the basis of past history, which allows it to optimize its response to (anticipated) future requests for resources of the same type.

Multiple administrative domains. The pipelined resource management architecture lends itself to distribution across multiple administrative domains because it schedules resources in a completely decentralized manner; all state information is carried with the query itself. Thus, it is easier to support distributed access control and usage policy enforcement within this framework. Moreover, the resource management pipeline facilitates a "system of systems" approach to scheduling: the pipeline can resolve a query down to, say, the level of a local resource management system, and then simply allow the local system to take over. (In this case, the "resources" within resource pools would be pointers to local resource management systems.) Currently, this capability is primarily used to allow the resource management pipeline to interoperate with grid middleware (Globus [9]) and cluster management systems (Condor [21], PBS [2], and Sun Grid Engine [25]).

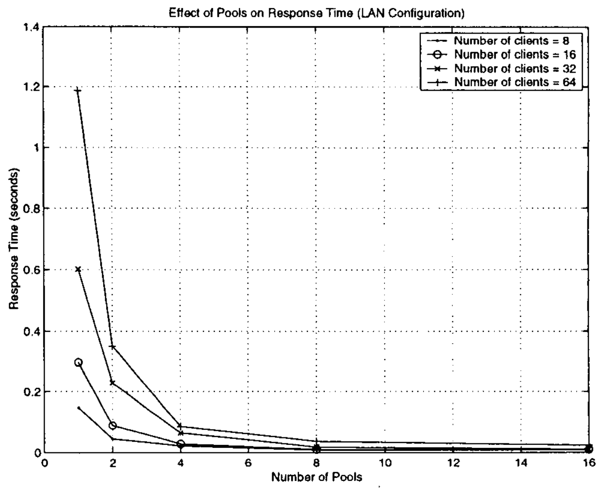

Figure 4. Effect of increasing the number of pools on response time in a local area network configuration. The experiment was conducted on a database of 3,200 machines, which were uniformly distributed across pools. Client queries were distributed randomly across pools.

7. Preliminary Evaluation

The results in this section are for an initial prototype of the active yellow pages service, and are based on synthetic workloads. All but one of the experiments described below were conducted within a local area network, with the clients running on Sun UltraSPARCs and the components of the ActYP service running on a 524MHz, 12-processor Alpha server. The remaining experiment was conducted with the clients running on an UltraSPARC at Purdue University (U.S.A.) and the components of the ActYP service running on an Alpha server at Universitat Politècnica de Catalunya (Spain).

The scalability of the resource management pipeline is primarily a consequence of the ability to replicate individual components of the pipeline. As an example, consider the benefit of using multiple pools. The effects of striping queries across increasing numbers of pools are shown in Figure 4 — note the reduction in response times with increasing numbers of pools. The results in Figure 4 are for a setup that is entirely within a local area network. When the clients and the ActYP service

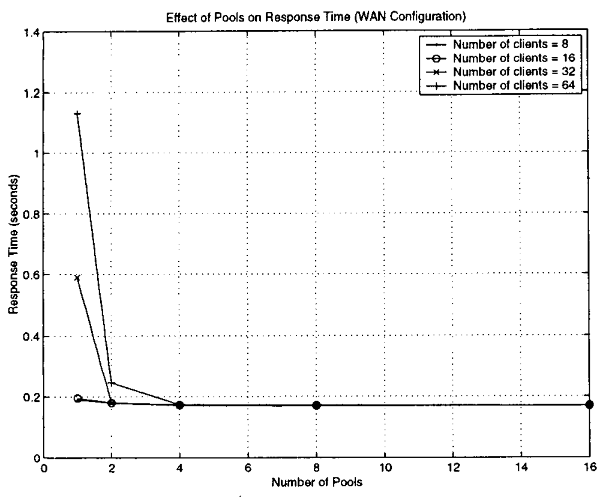

Figure 5. Effect of increasing the number of pools on response time in a wide area network configuration. The experiment was conducted on a database of 3,200 machines, which were uniformly distributed across pools. Client queries were distributed randomly across pools.

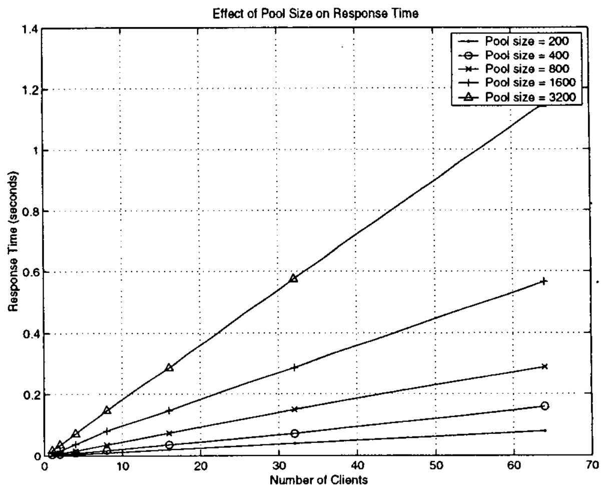

Figure 6. The response time as a function of the size of the pool. Clients continuously send queries to the ActYP service.

are distributed across a wide area network, multiple pools still help, but network latency limits the reduction in the response times (see Figure 5).

Scalability, in this context, also implies an ability to manage localized "hot spots". Such hot spots may happen, for example, in environments that have a large number of homogeneous resources — causing most resources to be aggregated in a single pool. Figure 6 shows what happens when the size of a pool grows. As expected, the response time degrades (the linear plots are simply a function of the linear search algorithms employed for scheduling). In such situations, pools could be *split*, allowing for concurrent searches whose results could then be aggregated. Figure 7 shows the results of such a solution — clearly, splitting improves the response time.

Another trigger for localized hot spots is when a large number of users request resources with the same specifications. This may happen, for example, when a large class is working on a lab or homework assignment. In such situations, it is necessary to improve the throughput of the resource management pipeline for a given set of resources. This can be accomplished by *replicating* pools, as shown in Figure 8. Replicated pools contain the same set of machines; scheduling integrity is maintained by introducing a instance-specific bias (e.g., instance 'i' of a given pool "prefers" every 'i'th machine in the pool).

8. Related Work

The PUNCH ActYP service has been designed with the PUNCH user base (students *and* researchers) in mind: the goal was to accommodate the needs of the relatively few specialized jobs without compromising the turn-around time for the large numbers of jobs with run-times in the range of a few seconds (see Figure 9). The service adapts its scheduling objectives according to observed resource requirements, and employs a non-preemptive, decentralized, sender-initiated resource management framework.

Cluster management systems such as Grid Engine [25], PBS [12] and DQS [11] typically utilize centralized schedulers. They accommodate jobs with diverse resource usage characteristics by employing multiple submit queues (e.g., one queue for short jobs; another for large ones). In contrast, ActYP utilizes a decentralized scheduler, and accommodates diverse jobs by routing them to appropriate nodes in its pipeline.

Opportunistic computing environments such as Condor [21] are designed to maximize the throughput for relatively large jobs. Condor employs a preemptive, centralized, receiver-initiated scheduling mechanism. The Globus resource management architecture [6, 10] is optimized for jobs that utilize highly-specialized resources and run for hours or days. It also supports

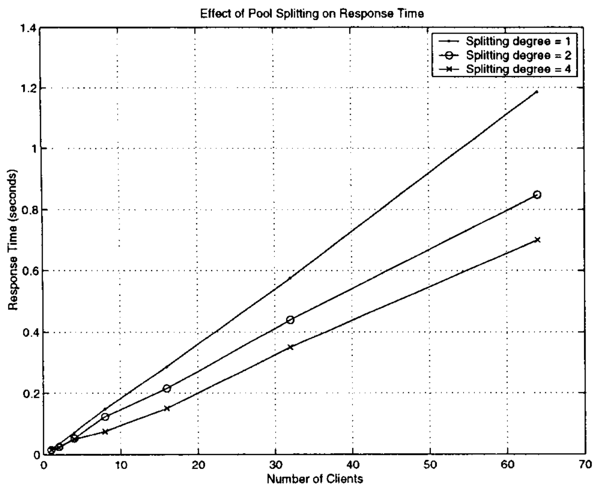

Figure 7. Effect of splitting on response time. The original pool consisted of 3,200 machines. It was split into 1) two pools with 1,600 machines each, and 2) four pools with 800 machines each.

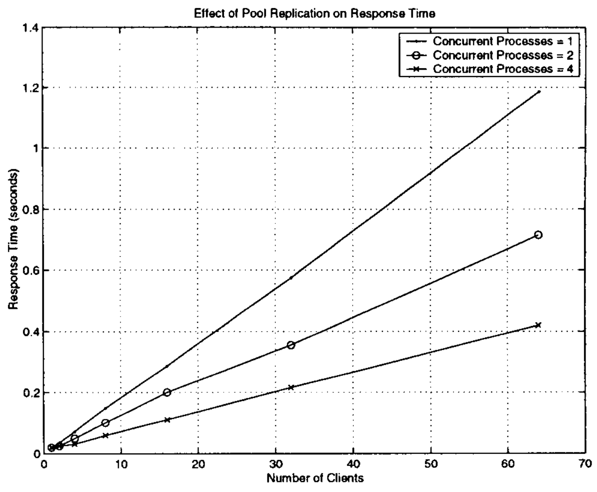

Figure 8. Effect of replication on response time. The pool contains 3,200 machines.

advance reservations and co-allocation of compute resources, neither of which are currently supported by ActYP. From a design objective standpoint, ActYP differs from Condor and Globus due to the need to support large numbers of short jobs and bursty submission profiles that are typical of academic environments.

Other approaches to resource management are the application-specific scheduling utilized by AppLeS [3] and the object-based scheduling utilized by Legion [4]. These approaches are not easily extensible to the PUNCH environment because of the large numbers of legacy applications utilized by PUNCH users.

9. Conclusions

This paper presented a novel, pipelined resource management architecture for computational grids. The design was based on two key realizations. One was that resource management involves a sequence of tasks that is best handled by a pipeline. The other realization was that static aggregation of resources for improved scheduling is inadequate in wide-area computing environments because the needs of users and jobs change with both, location and time. The described architecture addresses this problem by dynamically aggregating resources in a manner that continuously optimizes system response. This is accomplished by way of an active yellow pages directory that allows aggregation constraints to be (re)defined on the fly.

An initial prototype of the active yellow pages service has been deployed in the PUNCH network computing environment, and has been in operation for about one year. Experiences with the production PUNCH system and preliminary results from controlled experiments indicate that the prototype ActYP service performs well.

Ongoing work is aimed at expanding the functionality of the current prototype. In particular, the current implementation does not support composite queries, and employs manually configured tables for pool manager selection and resource pool creation. It also does not support delegation of queries from one pool manager to another. Future work will also focus on a more detailed evaluation of the effectiveness of the described approach in large, wide area environments.

Acknowledgements

This work was partially funded by the National Science Foundation under grants EEC-9700762, ECS-9809520, EIA-9872516, and EIA-9975275; by an academic reinvestment grant from Purdue University; and by the Ministry of Education of Spain (CICYT TIC-98/0511) and the European Center of Parallelism in Barcelona (CEPBA).

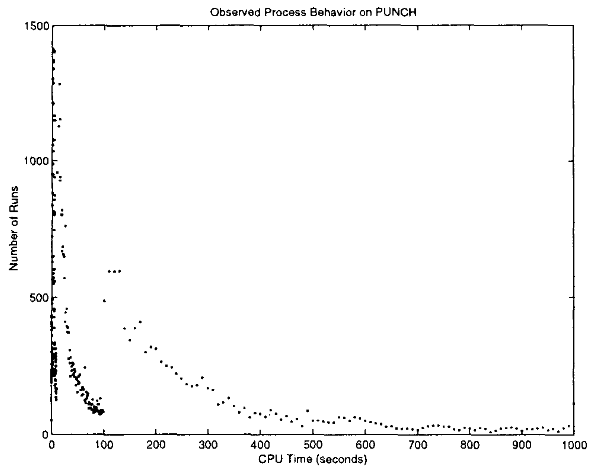

Figure 9. Distribution of measured CPU times for 236,222 PUNCH runs. The X- and Y-axes are truncated to show detail; observed CPU times extend out to more than 10^6 seconds, and the Y-axis extends to 19756 runs.

References

[1] S. A. Banawan and J. Zahorjan. Load sharing in heterogeneous queueing systems. In *Proceedings of the IEEE INFOCOM*, pages 731–739, 1989.

[2] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten. Portable Batch System: External reference specification. Technical report, MRJ Technology Solutions, November 1999.

[3] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of the 1996 Supercomputing Conference*, 1996.

[4] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. The legion resource management system. In *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, San Juan, Puerto Rico, April 1998. Held in conjunction with the International Parallel and Distributed Processing Symposium.

[5] D. E. Comer. *Internetworking with TCP/IP - Volume I: Principles, Protocols, and Architecture.* Prentice-Hall, 1995.

[6] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Teucke. A resource management architecture for metacomputing systems. In *Proceedings of the Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, 1998. Held in conjunction with the International Parallel and Distributed Processing Symposium.

[7] R. J. Figueiredo, N. H. Kapadia, and J. A. B. Fortes. The PUNCH virtual file system: Seamless access to decentralized storage services in a computational grid. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC'01)*, San Francisco, California, August 2001.

[8] S. Fitzgerald, I. Foster, C. Kesselman, G. v. Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing (HPDC'97)*, pages 365–375, 1997.

[9] I. Foster and C. Kesselman. The Globus project: A status report. In *Proceedings of the 1998 Heterogeneous Computing Workshop (HCW'98)*, pages 4–18, 1998.

[10] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service*, London, U.K., 1999.

[11] T. P. Green and J. Synder. DQS, a distributed queueing system. Technical report, Florida State University, March 1993.

[12] R. L. Henderson and D. Tweten. Portable batch system: Requirement specification. Technical report, NAS Systems Division, NASA Ames Research Center, August 1998.

[13] H. Kameda, J. Li, C. Kim, and Y. Zhang. *Optimal Load Balancing in Distributed Computer Systems.* Springer, 1997.

[14] 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 Workshop on Advances in Parallel and Distributed Systems (APADS)*, pages 372–377, West Lafayette, Indiana, October 1998. IEEE Computer Society.

[15] N. H. Kapadia, R. J. O. Figueiredo, and J. A. B. Fortes. PUNCH: Web portal for running tools. *IEEE Micro*, pages 38–47, May-June 2000.

[16] N. H. Kapadia, R. J. O. Figueiredo, and J. A. B. Fortes. Enhancing the scalability and usability of computational grids via logical user accounts and virtual file systems. In *Proceedings of the Heterogeneous Computing Workshop (HCW) at the International Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco, California, April 2001.

[17] N. H. Kapadia and J. A. B. Fortes. PUNCH: An architecture for web-enabled wide-area network-computing. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 2(2):153–164, September 1999. In special issue on High Performance Distributed Computing.

[18] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley. Predictive application-performance modeling in a computational grid environment. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC'99)*, pages 47–54, Redondo Beach, California, August 1999.

[19] N. H. Kapadia, J. A. B. Fortes, and M. S. Lundstrom. The Purdue University Network-Computing Hubs: Running unmodified simulation tools via the WWW. *ACM Transactions on Modeling and Computer Simulation (TOMACS),* 10(1):39–57, January 2000. In special issue on Web-based Modeling and Simulation.

[20] P. Krueger and M. Livny. The diverse objectives of distributed scheduling policies. In *Proceedings of the 7th IEEE International Conference on Distributed Computing Systems*, pages 242–249, 1987.

[21] 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*, pages 104–111, June 1988.

[22] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC'98)*, pages 140–146, Chicago, Illinois, July 1998.

[23] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, January-February 1998.

[24] S. Shenker and A. Weinrib. The optimal control of heterogeneous queueing systems: A paradigm for load-sharing and routing. *IEEE Transactions on Computers*, 38(12):1724–1735, 1989.

[25] Sun Grid Engine. Web site at www.sun.com/gridware.