A Mechanism for the Dynamic Construction of Clusters Using Active Networks

*Peter Graham** and Rajendra Singh*

Parallel and Distributed Computing Laboratory
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2

{rajendra,pgraham}@cs.umanitoba.ca

February 5, 2001

Abstract

In this paper, we describe an active networks architecture for dynamically constructing [clusters of] clusters in response to user requests for computational service. Active network nodes (i.e. routers) match requests for computational service to offers of such service using recent access and usage pattern information to construct on-demand, wide-area clusters. By combining appropriate offers of service with knowledge of current (as well as predictions of future) processor load and network conditions, the system dynamically constructs clusters to cost-effectively meet user needs. The active networks approach exploits the fact that network devices are in an ideal position to decide which resources should be combined to build clusters in response to particular requests. This approach offers advantages in terms of anonymity of service (systems are unaware of one anothers' existence), scalability (scheduling is distributed across many network processors), fault tolerance (service is distributed so failures are recoverable and/or localized) and automatic localization (clusters are created as close to the requestor as possible).

Keywords Active Networks, Clusters, Computational Grids, High Performance Computing, Resource Discovery and Allocation.

1 Introduction

The use of cluster-based systems to solve computationally challenging problems has increased rapidly with the deployment of high speed, switched, local area networks

*This research was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Grant OGP-0194227.

(LANs). Such networks make clusters a serious alternative to tightly coupled multiprocessors for the solution of many large-scale scientific and engineering problems. With the advent of high speed (gigabit per second) wide-area networks, switched LANs, and the increasing deployment of fibre in metropolitan areas, it is now possible to consider the construction of clusters of significantly larger scale and across much wider areas than is now common.

Clusters make an appealing architectural base for the construction of computational grids for many reasons. First, they are a very flexible and general purpose platform on which to build a wide range of high performance applications. Clusters can also take many forms - consisting of multiple levels (clusters of clusters), being of various "shapes" (based on interconnect topology), and having various node types ranging from simple desktop machines to multiprocessors. Finally, with the increasing deployment of fast, switched LANs interconnecting desktop machines there is a huge pool of available cycles that can be used for high performance, not just high throughput, computing. The construction of large-scale clusters from existing computing resources is an extremely cost-effective way to acquire additional compute power.

For the reasons just described, we are actively researching techniques that will permit the *simple and reliable* construction of wide-area clusters that are effective to use in solving a wide range of computational problems. Building large-scale cluster systems using existing technology is, however, challenging. Our research seeks to more tightly integrate computation and communication in wide area clusters through the use of "Active Networks" to simplify cluster construction.

Active networks allow computation to be dynamically migrated into the network itself and this offers many potential benefits. Active networks advocates argue that certain computations are more naturally performed in the network where they can exploit their location and/or information about network load to optimize their processing. We believe that this argument also holds when considering the scheduling of work onto processors within a cluster-based grid environment.

A common problem in distributed algorithms generally (and grid management algorithms in particular) is the need to know the identities of other machines with whom you must communicate. This requirement leads to the need for centralized location

services (name lookup, etc.) at “well-known addresses”. Any such centralized service represents a single point of failure and the impact of such a failure grows with the size of the system that is dependent on the service for correct function. Centralized services also tend to lead to bottlenecks especially as the size of a system increases. Not only can the processor providing the service become overloaded, but so too can the network around it. As a result, such centralized systems do not scale well.

The active networks based approach described in this paper addresses these problems. It exploits the fact that its code runs “in-network” to eliminate the need for processes to know service addresses and to provide better fault tolerance and more efficient processor selection. Our system *dynamically* constructs clusters from available processors to meet the needs of users throughout the grid.

1.1 Organization

The rest of this paper is organized as follows. In Section 2 background material and work related to that presented in this paper is reviewed. Section 3 discusses the problem and our assumed environment. Our proposed solution is presented in Section 4 and we discuss the status of our prototype implementation in Section 5. We present our conclusions and discuss areas for future research in Section 6.

2 Related Work

2.1 Cluster Computing

With the development of relatively low-cost, high performance local area networks, doing parallel computing with collections of workstations has become practical for many computational problems. While still lagging behind more tightly coupled parallel machines in terms of raw performance, “clusters” [8, 9] offer surprisingly good performance at extremely low cost. A number of different sorts of cluster architectures have recently been proposed ranging from shared networks of workstations (e.g. NOW [2]) to dedicated cluster machines as in the Beowulf [7] approach.

Generally, cluster programming is done using message passing systems such as

PVM [17], MPI [20] and HPVM [11] although software-based Distributed Shared Memory (DSM) systems (e.g. [31, 23, 24, 27]) have also been developed in an effort to simplify programming. At the implementation level, however, both message based and DSM systems depend on *a-priori* knowledge of the identities of the machines that form a computing cluster[1]. The work presented in this paper dynamically selects the nodes that will form a cluster and it can be easily integrated with systems like MPI.

In a wide-area distributed environment, there is the potential for significant disparity in the speeds of the network between specific machines in a cluster. Currently the management of the resulting non-uniform messaging times is the responsibility of the programmer and as a result clustering is normally limited to local area, homogenous network environments. In future cluster programming systems the management of non-uniform messaging will be at least partially automated. Our technique is also capable of providing the information needed to support this added functionality.

2.2 Computational Grids

Much interest has been generated recently in constructing national-scale computing infrastructure by linking together machines at many locations to allow the development of very large scale distributed computing systems. By analogy with the power grid, such systems have come to be known as "computational grids" [16]. Building on experience gained with earlier meta-computer systems (e.g. [6, 26]), grids offer the potential for access to massive computational capacity that will allow researchers to solve the most challenging computational problems. They also offer much broader connectivity between machines that will enable innovative applications that exploit the resources and data provided by the constituent machines.

A variety of software has been developed that focuses on the construction of grids both generally and for specific applications. General purpose grid software systems (the focus of this paper) strive to provide environments or tools that can be used by application developers to build large scale distributed systems. Challenges to be met

[1]For example, although MPI programs are independent of the machines they are running on, the MPI runtime system must know machine names to map them to the abstract "nodes" referred to in the programs.

by such software arise largely due to the size and shared nature of the systems being built. Issues such as performance scalability, fault tolerance, and management of both heterogeneity and autonomy complicate the design of grid systems.

Examples of systems used to support grid construction include Legion [19] and Globus [15]. Legion provides an *object-based* Grid software development *environment*. By exploiting objects, Legion achieves benefits including self-management (via a meta-class mechanism), uniformity across heterogenous platforms, and a good deal of transparency. This is achieved, however, at the price of forcing application developers to learn and use a new system.

Unlike Legion, Globus adopts a tool-kit approach where developers are not required to develop code to meet a particular API. Instead, Globus provides what is referred to as a "bag of services" that developers may pick and choose from to best meet individual application needs. The Globus approach offers faster development of applications from existing code and is very flexible. This latter point is of particular importance, now, when the best choices for grid design have yet to be clearly identified.

Unlike both Legion and Globus, the Condor system [25] specializes in using collections of idle workstations to provide high-throughput computing (i.e. efficient execution of large collections of independent tasks). Our research exploits ideas from Condor and combines them with other concepts to define a grid consisting of dynamically created, multi-level clusters built, initially, from desktop machines and departmental servers. The focus of our work, however, is not limited to high-throughput computing.

2.3 Active Networks

Active networks [10, 29, 32, 34, 21, 22] are a novel approach to network architecture in which customized (possibly application specific) programs can be executed within the network itself. By introducing the ability to execute user-defined programs, the network is no longer restricted to packet delivery alone and this creates opportunities for optimized solutions to a number of problems that arise in distributed systems.

Active packets can be encapsulated within an IP packet using the Active Network Encapsulation Protocol (ANEP) [1] thereby making them transparent to network nodes

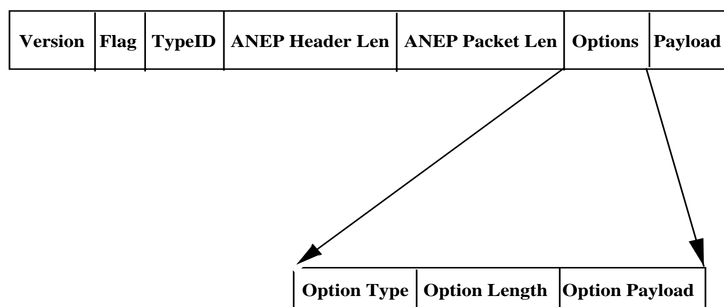

Figure 1: Format of an ANEP header

that do not support active packets. This will allow the gradual deployment of active routing elements into the Internet while maintaining compatibility with non-active elements. Non-active network nodes simply ignore the active fields (since they are invisible to them). The Option Type field in an ANEP header (see Figure 1) determines how active nodes should handle a given active packet type. If a service corresponding to the specified type does not exist at an active node it must be asynchronously loaded.

Code distribution techniques allow active services to be automatically and dynamically transferred to router nodes as they are needed. With a code distribution mechanism, we are no longer obligated to deploy all (or any) services to network nodes in advance. Thus, nodes not requiring certain services do not incur the overhead of storing their code. Dynamic code distribution makes it is easy to develop, maintain, and upgrade new network services and to spread them throughout an active network. ANTS [33], a prototype active network architecture from MIT, provides on-demand, Java-based code distribution.

Active networks technology is generating significant interest in the network research community. This interest is reflected in the development of `anetd` [12] (an active version of Unix's `inetd` server) as part of the ABONE active networks research testbed. In this paper we propose the use of active networks technology (initially using ANTS) to integrate the network into a cluster-based grid environment. We will show that specific benefits over existing approaches are achievable by using active networks.

3 The Problem and Environment

Many researchers would benefit from the availability of computational resources greater than what they have at their disposal. Unfortunately, large-scale, high-performance computing facilities are often cost prohibitive. To address this problem, researchers are exploring the use of collections of networked machines. In most organizations there is a wealth of unused compute cycles available and using them is very cost-effective. This idea can also be expanded to larger scales. Unfortunately, the implementation of such "clustered" systems introduces new problems that must be solved. This paper addresses the problem of locating suitable machines to run work on by matching offers of compute service to compute requirements. This matching is done "in-network" using active networks technology.

Thus, we are proposing the use of active networks for dynamically creating grids structured as hierarchies of clusters. The framework that we have designed for doing this is tailored to a specific (hierarchical) networking environment. Despite this, the framework is more generally applicable and could be readily applied in less structured network environments. By basing our framework on active networks, the flexibility that is needed to support different environments can easily be made available.

3.1 The Computing/Network Environment

Our prototype framework was designed to run in a test environment that will consist of machines in labs distributed across several Canadian universities. All potential cluster nodes are x86-based Linux machines[2] that are connected together and to other labs in each University using switched fast Ethernet (or better). The Universities are connected using CA*Net3, an optical Dense Wave Division Multiplexed (DWDM) network capable of carrying up to 40 Gbps of traffic.

Our initial, assumed network topology is a shallow tree (see Figure 2). We further assume that network nodes are, in some sense, aware of their "position" in the tree structure. That is, they are aware of which other node lies beneath/above them in the

[2] While we assume homogenous processors at this early stage of the research, there is nothing intrinsic to the work presented that precludes the construction of clusters consisting of heterogenous machines.

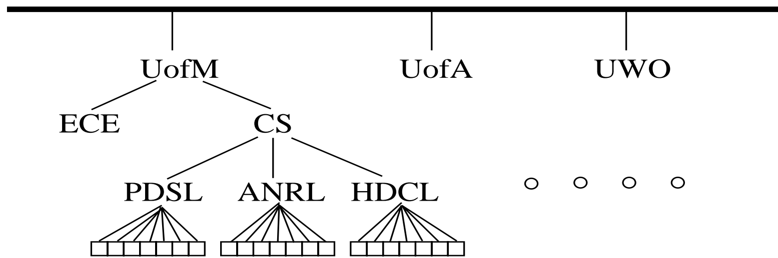

Figure 2: Assumed Network Environment

tree. This assumption makes the discussion of our algorithms simpler but is not necessary to use them. If network position information is not available, our algorithms can still be applied using a form of restricted flooding. Thus, our work can be generalized to larger and less structured networks.

The computing environment we assume for our prototype simplifies the problem of dynamically creating clusters in at least two ways. First, since our prototype environment assumes a single compute platform, we do not have to address heterogeneity issues. Second, thanks to the cooperation of the university labs involved, autonomy concerns are not an issue. In a "real-world" grid, such assumptions would not be possible but because these issues are largely orthogonal to the work presented here they will not be discussed.

Finally, we also assume that the "active engine" in each active router (that executes the active networks code) is separated from the normal packet routing engine. This means that traditional routing is not affected by active engine failures. It also allows for more aggressive in-network computation since significant computing resources can then be added to existing routers [18]. This is important because it will allow more complex grid allocation algorithms than the one discussed in this paper.

3.2 Assumed Job Characteristics

Another important factor affecting the work presented in this paper is our assumptions about the characteristics of the computations that will be done using our proposed sys-

tem. Our current focus is on long running, batch type jobs. Further we do not consider the issue of checkpointing and restarting computations. While this is clearly an important component in a reliable computing environment for long-running computations it is beyond the scope of this paper to consider.

The importance of assuming long running jobs is two-fold for our research. First, it allows us to tolerate a certain amount of overhead in our resource allocation algorithm. This is important since selecting a good cluster may require some effort. Further, in an environment where there may be many competing requests for cluster formation, it may not be possible to successfully allocate the resources in the first "candidate" cluster. In this case the selection process may have to be repeated. The second reason long-running jobs are important is that it allow us to consider scheduling in fairly coarse-grained scheduling intervals. This decreases the problem size for our in-network algorithms – where limited computation overhead is normally a requirement.

3.3 Motivation for Dynamic Clusters

Currently, most cluster use is limited to statically configured systems running on LANs. Users are restricted to running only on machines designated for cluster use. There are often many other compute cycles available on machines that are "near enough" to be useful but which may not be a part of the user's statically-defined cluster. Further, with recent improvements in installed network infrastructure (i.e. the change from shared 10Mbps to switched 100Mbps Ethernet), it is now possible to build clusters beyond the boundaries of a single LAN (e.g. a "flock of Condors" [13]).

Being constrained to static cluster construction limits users' access to compute capacity. For example, one of the author's cluster system is heavily used while a nearby lab housing machines with roughly the same computational and nearly the same communication capacity is commonly underutilized. A system that dynamically constructs clusters from arbitrary, underutilized machines could exploit the alternate facility to construct a larger and/or more effective cluster. An additional benefit of this approach is improved overall resource utilization. Computational load is effectively balanced across all available machines using our system since lightly loaded machines are se-

lected for cluster construction before those that are more heavily loaded.

Dynamically constructing a cluster, first involves locating machines from which to form the cluster and determining their characteristics and the characteristics of the network(s) connecting them. This is often referred to as "resource discovery". A system such as the Network Weather Service (NWS) [35] can be used to perform resource discovery. The NWS collects information from resource monitors that have been placed on potential resources and then provides this information to any scheduler interested in using it.

Once computational resources are known, the needs of computational requests must be matched with the capabilities of potential cluster nodes to select an appropriate dynamic cluster. This is referred to as "resource allocation" and is essentially a form of admission scheduling. Since the capabilities of each potential cluster will vary over time this matching process must be, at least partially, predictive. Using past usage patterns for this purpose not only allows prediction of future usage but is also cheaper and often more accurate than frequent, on-going monitoring.

A straightforward, host-based solution to matching service offers to requests would centralize the decision making process at one site. The selected site, unfortunately, becomes a single point of failure and both the site and surrounding network are prone to overloading as the size of the system scales (a fundamental requirement for a national grid). Such a system is also prone to catastrophic failure (which is unacceptable). An alternate approach might distribute the decision making process across a number of host nodes thereby improving fault tolerance and scalability. Such an implementation, however, is complicated by the need for each potential cluster node to have knowledge of the locations of the scheduling hosts. Further, recovering from the failure of one scheduling host is extremely complex and costly (in terms of network communications).

Using active networks to dynamically construct clusters solves both these problems. All messages related to cluster construction are "blind" (i.e. sent into the network without a destination address) so there is no need for participating machines to know the locations of any system components. The failure of any scheduling component is also completely transparent to system users.

4 An Active Networks Framework

We use active networks in two ways: to gather information for the cluster construction process and to implement the dynamic construction of clusters by matching service requests to (possibly combined) offers of service that collectively meet service needs. The active protocols that perform these functions obtain their needed information from code that is dynamically deployed on both host machines and network routers. In what follows, we describe a relatively simple scheme for describing machine capabilities and job requirements that is used in our prototype. The exact information exchanged and algorithms applied to do resource allocation are not important to the architecture we are proposing in this paper. As such, no attempt is made to provide *details* on how information used in the allocation process is derived or how it is applied to make allocation decisions. The interested reader is referred to work on the Network Weather Service [35] which includes details on how that system gathers load information and to work on market based scheduling [14] as an example of how such information can be used.

4.1 Monitoring Network Usage

To construct an efficient cluster, the capacity of both the consitutent machines and the network interconnecting them must be considered. We propose using active networks to monitor and record information about network usage patterns over time. The goal of our active monitoring protocol will be to extract information about network capacities and usage patterns that will allow prediction of future network use (for the time when that network may be used to form a cluster).

Any protocol for network monitoring (active or not) must be non-intrusive, low-overhead, and as accurate as possible. Further, since the *exact* information that will be needed to support effective cluster construction is still largely an open research question, the protocol should also be flexible. The use of active networks helps to satisfy all these requirements.

Each active routing node in the network will run monitoring code to gather the needed data. Devices at the edge of networks which connect to host machines can de-

rive their information through periodic "ping" and "echo" type messages to assess the latency and bandwidth to/from host machines. Edge network nodes will then provide summary information to their parent nodes in the hierarchy. Those parent nodes will combine that information with data about the network link to the node supplying the information. This process can then be repeated hierarchically to disseminate information throughout the network. A special case exists at the "root" of our assumed hierarchical network environment but it is a simple extension of the basic technique just described.

Each active node in the network will also extract information from the routing code (e.g. number, size and type of packets passing through the node) to build temporal, summary usage information. This information can be used to determine usage patterns that can then be applied to predict future usage. This information will ultimately be used to help decide if a potential cluster will meet the communication needs of a particular application.

The extracted information must be stored compactly since there is, typically, limited space available in each router's "soft store" (i.e. local memory). While space is limited, leaving usage information in the soft store has the benefit that the active cluster construction code does not need to re-load the information from a host system nor does it need to know the address of such a host. The tradeoff between the accuracy of stored information (based on its size) and the cost of loading larger, more accurate information will, we believe, be an interesting are of future research.

4.2 Offers of Computational Service

In addition to network usage information, there are two other inputs to the cluster construction process: offers of, and requests for, service. Machines must announce their willingness to provide service and users must request cluster service to meet the requirements of their jobs.

Individual machines may provide offers of service based on their current and/or predicted usage which they inject blindly into the network. Active network code will ultimately combine such offers of service with information from the previously described active monitoring protocol to construct appropriate clusters to satisfy applica-

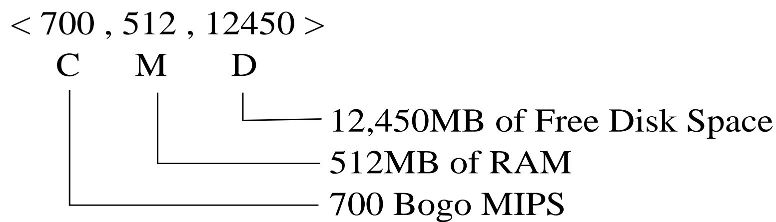

Figure 3: Example $< C, M, D >$ Triple

tion needs.

Offers of computational service must provide information about the machines themselves as well as information about when such offers of service hold. In our prototype implementation, information about individual machines at some instant in time is captured using a triple $< C, M, D >$ (representing normalized estimates of Compute, Memory and available Disk capacity – See Figure 3). Since there are often regular usage patterns for individual machines in a university environment (the target of our prototype implementation) we extend this information to predict resource availability over a time period during which the usage patterns are likely to hold (in the prototype this is 1 week). An offer of service is thus represented by a time-varying vector of triples $< C, M, D >^*$. Each such vector is arranged so the first entry always corresponds to the first scheduling interval in a week (e.g. beginning midnight, Monday morning). Dividing a week into scheduling intervals of 15 minutes results in a sequence of 672 $< C, M, D >$ triples. Each value in a triple is represented by a 16 bit unsigned integer so a week of predicted availability information for a single machine is less than 4K in size.

Machine usage information is derived by daemon processes downloaded to, and running on, the machines. These daemons periodically inject their information blindly into the network. Daemons on machines also track their own usage patterns and only announce *noticeable* changes when they are detected. Thus, the frequency of injection is typically quite low and the overhead on the active routers is small. As machines are increasingly used in clusters and new (hopefully regular) usage patterns emerge, the

patterns are *automatically* disseminated into the network and the cluster construction process can adapt accordingly.

Ultimately, the usage information provided to the active network nodes is only predictive and may not reflect the actual state of any given machine at a particular instant in time. To help minimize the potential for unexpected variance between the predicted load and the actual load, our system propogates resource allocation information up and down the network hieararchy so that other active nodes are aware of resource commitments made by other nodes. This information is, hoever, not sent to the processor daemons which only track *actual* machine usage.[3]

If, at some time, a candidate cluster is selected that offers *just enough* capacity for a given job and there is unexpected load on some of the machines in the cluster then performance will suffer. We view this as an acceptable (and arguably unavoidable) situation. If the machines available to construct clusters are being very heavily used then jobs will have to share resources and completion times will increase. This is no different from overloading a single or multiprocessor machine and simply indicates the need to increase the available resources.

When the use of the machines is not high there will be several candidate clusters to choose from and one will be selected that is most likely to ensure good performance. Further, since the prediction mechanism is self-adaptive and the typical jobs running on such clusters are long-lived, the information used in scheduling is likely to result in a reasonable assignment of work to machines in most cases.

4.3 Requests for Computational Service

A machine that wishes to request computational service does so by constructing and injecting a *computational schema* into the network. A schema describes the characteristics of the computation so the active cluster construction algorithm can map available resources to the computation and thereby construct a cluster for it. Generally, a schema must describe the key aspects of the computation including such things as the number and capabilities of the machines needed, the expected running time of the job on

[3] More will be said about the processing of resource allocations in a later subsection.

each machine, as well as information about the communications expected between the various machines (volume and frequency of communication, latency sensitivity, etc.).

Such a schema can be represented using a graph structure with nodes for processes and edges between nodes for communicating processes. Nodes can be attributed with the minimum required characteristics of the machines that will run the corresponding processes (in the form of "desired" $< C, M, D >$ information) and the edges can be attributed with estimates of the frequency and volume of communication that will take place between the machines. Further, arbitrarily complex sub-graphs can be created to represent program components with specific computation and communication needs and then embedded as nodes in other graphs. This mechanism provides the ability to describe programs that will run best on systems composed of multiple parallel machines (i.e. metacomputing).

Such a general schema structure leads to a complex resource allocation process involving the identification of groups of processes that have the heaviest inter-process communications and mapping them to locally-connected groups of processors subject to the constraint that the selected processors meet the minimum requirements of the processes for the duration of the job.

The complexity of resource allocation must be decreased for an in-network solution that uses existing active network architectures and this impacts the schema structure. In our target environment, it is possible to reduce scheduling complexity in two ways by exploiting the assumed hierarchical network environment. First, the graph can be restricted to being tree structured. This reduces the worst-case number of communicating machines from $O(N^2)$ to $O(N)$. Second, we can allow a schema to include "compound" nodes that represent groups of tightly-coupled processes that are to be scheduled together as a unit (onto machines connected by a switch). This allows the cluster construction algorithm to ignore the details of communications between machines in the compound node. As active node architectures become available that support more aggressive computation, such constraints on the resource allocation algorithms may be relaxed.

4.4 Active Resource Allocation/Cluster-Construction

Cluster construction must match requirements in a computational schema to available resources for the expected duration of the computation. Information about *individual* process requirements can be matched directly against the capabilities of machines in their $< C, M, D >^*$ vectors. Information about the expected communication between machines can also be matched against the collected network capacity and usage information to select machines so as to minimize communication delays.

When new $< C, M, D >$ values are injected into the network by daemons running on specific machines, the active routers (A.R.s) that receive them store the packet's information in their soft stores (as a part of that machine's $< C, M, D >^*$ vector) and pass the information up to any A.R.s above them. In this way, the available computational capacity offered by a machine is disseminated throughout the network and becomes available for use in cluster formation anywhere in the network. Higher level A.R.s always receive the $< C, M, D >$ information of *all* machines beneath them even if there are one or more intervening levels of routers. Note also that the failure of an allocation algorithm at an A.R. does not prevent cluster construction. It simply forces it to occur at a higher level A.R..

Each A.R. maintains a collection of $< C, M, D >^*$ vectors, one per machine "underneath" it. Each such vector is maintained as a circular queue of $< C, M, D >$ values (see Figure 4). The first element of each such vector reflects expected capacity at the corresponding machine in the first time/scheduling interval. When processing past the end of a circular queue, the allocation algorithm simply continues at the beginning of the queue. This situation reflects a computation that spans from one scheduling period (e.g. 1 week in the prototype) to the next.

To save computational overhead during allocation processing, each A.R. periodically[4] constructs a set of Resource Availability lists (`RALists`) from the $< C, M, D >^*$ queues. `RALists` describe candidate clusters that are/will-be available for use. Such lists are built for specific, likely cluster sizes, compute durations, and start intervals. In our prototype, candidate clusters are restricted to being of sizes that are powers of

[4] Hopefully at times of low load.

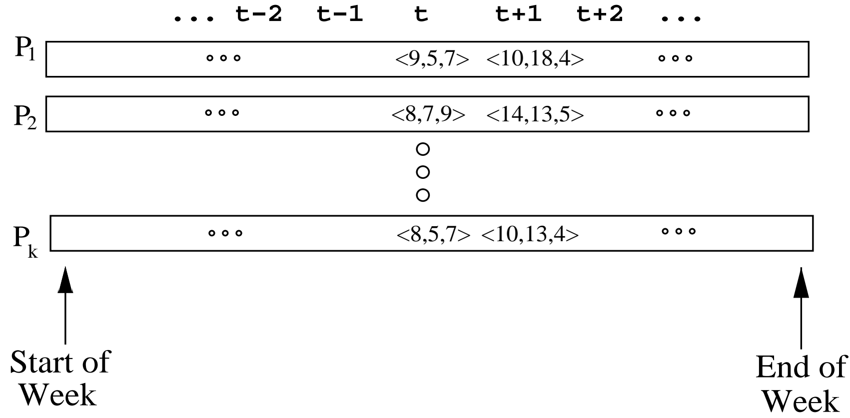

Figure 4: An A.R.'s set of Circular $< C, M, D >^*$ Queues

two (up to the number of processors available beneath the router constructing the lists) and durations are restricted to being powers of two (up to the number of scheduling intervals in a week).

After some initial system use, a set of reasonable `RALists` will exist and these only *have* to be updated (using information derived from new $< C, M, D >^*$ vectors) when a request arrives for a cluster that will exist long enough to "bump into" stale `RAList` information. This means that `RAList` updates can be done either on-demand or in an otherwise lazy fashion to minimize detrimental effects on A.R. performance. It also means that A.R.s can discard parts of the `RALists` to save space if necessary and re-generate them from the more compact $< C, M, D >^*$ information as needed.

The construction of a simple `RAList` is illustrated in Figure 5. In this example, there are two machines P_1 and P_2 and the algorithm is forming an `RAList` for a cluster formed from the two machines, P_{1-2} that is available from the current time, t, for two scheduling intervals. The *conservative* `RAList` values are simply the minimums of the corresponding $< C, M, D >$ triples.

The total amount of information stored by an A.R. is determined by the number of machines it can schedule on and the number of scheduling intervals. As a result,

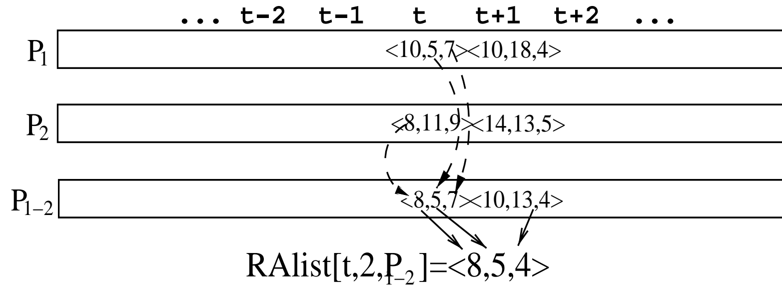

Figure 5: Construction of an `RAList`

A.R.s near the leaves of the network hierarchy store much less information than A.R.s near the top (which see more machines). In a realistic grid environment, the limited storage capacity of high level A.R.s could limit scalability. To address this it is also possible to pass summary `RALists` up to higher level routers instead of forwarding all $< C, M, D >$ messages.

In the prototype, `RALists` are sorted in each A.R. by starting time interval, then duration, then number of processors, then memory capacity, and finally by disk capacity. Precomputing and sorting the lists minimizes A.R. overhead when requests for service are made.

When computational schemas are injected into the network the receiving A.R. attempts to provide service locally but if it finds it has insufficient available resources (i.e. no suitable `RAList` it forwards the schema up to a higher-level A.R. which will have knowledge of, and hence access to, additional machines. Using this approach requests for service are always satisfied as close as possible to the machine that injected the request. This provides a natural sort of fairness to the allocation process. Only when "local" resources are unavailable or the schema describes an exceptionally large computation are distant compute resources enlisted to solve the problem. This will also be useful when running parallel jobs that produce results "on-the-fly" which must be delivered to the machine that created the job since network transmission distance will be minimized.

In the prototype, information on network capacity and usage is collected but is currently not used. Correspondingly, computational schemas are assumed to be flat. This was done because:

- it simplified the prototype implementation,
- given the high-bandwidth connections available and the comparatively coarse granularity of the code we expect to run there is little need to focus on communications issues, and
- none of the programming tools we have currently support hierarchical clusters.

At any given point in time an A.R. in the prototype has, in its soft store, the sorted `RALists`. A flat schema requesting a cluster consists of some $N < C', M', D' >$ triples and a time duration t expressed as a number of scheduling intervals over which the cluster is needed. The A.R. attempts to find N machines with $< C, M, D >$ values that can provide the needed capacities ($< C', M', D' >$). This is done by constructing a conservative estimate of resource *requirements* for the duration of the computation in a fashion similar to how the `RALists` were created (except that instead of selecting minimum capacities, we select maximumum requirements). The `RALists` are then searched to locate all candidate clusters that could satisfy the request's requirements. One of these is selected at random as the cluster to be used for the computation. Selecting a candidate randomly helps to ensure reasonable load distribution over candidate clusters without the need to maintain additional state in the A.R.s.

Once a cluster has been constructed, the set of selected machines is returned to the service requestor. This information can then be used by the requestor to distribute the needed code, data, and control information to the various machines and to initiate processing. The actual execution of user jobs is not a part of this research. Thus issues such as process distribution mechanisms and the [co-]scheduling (e.g. [28, 4, 30]) of processes across cluster nodes are not discussed.

An allocation of a set of machines to create a dynamic cluster that will execute a job at some future time represents a resource commitment. It is important to distribute such commitment information to all A.R.s that could schedule work onto a processor

in the same set of machines. If this is not done then, other A.R.s will make allocation decisions based solely on past usage information and not considering commitments that have also been made. The result, as described earlier, is a greater likelihood of over-commitment and corresponding poor performance.

When the description of a selected cluster is sent to the requesting processor, it may pass through some A.R.s which sit between the allocating A.R. and the requesting machine. Such routers could not service the request (due to inadequate resource availability at "their level") but are potentially capable of scheduling other work onto some of the processors allocated by the higher level A.R.. These lower-level A.R.s can easily "snoop" on allocation result messages to learn of resource commitments. Such commitments are easily reflected in each A.R.'s queue of $< C, M, D >^*$ vectors by simply subtracting the committed resources from those recorded as being available in the vectors.

The process just described ensures that lower-level A.R.s receive resource commitment information. It is also necessary for higher-level routers (up to the "root" of the tree) to receive allocation information since they too may schedule work onto processors which have been allocated to other jobs. To address this problem, resource commitments are also sent "up" the tree to the root (albeit as special resource-commitment messages not as responses to resource allocation requests). The higher level A.R.s process such messages in the same way lower-level A.R.s process allocation result messages.

5 Implementation Status

We are currently constructing a "proof of concept" implementation based on the ANTS toolkit [33] and ANETD [12] using Linux systems as both the hosts and the routers. At this stage, system components are being tested exclusively within the University of Manitoba. Many system components are finished or nearing completion (including the host daemon and the active network monitoring code). Other parts are still being developed and tested (e.g. the actual cluster construction code).

Following the successful integration and testing of the components we plan to exer-

cise the system locally to assess the limitations of the ANTS implementation and then re-implement for improved efficiency if necessary. Before actually seeking to deploy the system in a live wide area network, we will run it in a testbed environment where we can use software to introduce delays that will simulate the variation in latencies experienced in wide area communications. Our final goal is to actually use the system across universities via a real, high bandwidth, wide area network.

6 Conclusions and Future Work

In this paper we have presented a novel active networks based framework for dynamically creating cluster-based grids to solve large scale computational problems. We have argued that involving the network in the creation of such clusters provides explicit benefits over other techniques that include anonymity of the participants, better scalability and fault tolerance, simplified construction of clusters and localization of service. We have also discussed techniques for using active networks to gather information about network activity and to dynamically match offers of computational service to requests for such service and have described a prototype for a realistic but limited environment.

The research presented in this paper may be extended in many ways. The first extension we will pursue is having the cluster selection algorithm incorporate the existing information about communication requirements and network capacities. This will be necessary to allow the framework to scale to the national level. Other specific improvements to the cluster selection algorithm we are exploring are the application of combinations of past usage patterns at different time scales, tracking the number of jobs submitted per machine to more economically recognize changes in user behaviour, and relaxing the current constraint requiring “flat” schemas.

We are also in the process of developing a technique for managing processes once they have been allocated to specific clusters. By extending “resource containers” [5] across multiple machines (as with Cluster Reserves [3]) and adding support for coscheduling and QoS parameters, we hope to create a flexible and powerful grid resource management architecture that is compatible with the active networks-based resource discovery and allocation techniques described in this paper.

References

[1] D. Scott Alexander, Bob Braden, and Carl A. Gunter. Active Network Encapsulation Protocol (ANEP). Active Networks Group, RFC Draft, July 1997.

[2] T. Anderson, D. Culler, and D. Patterson. A Case for NOW. *IEEE Micro*, 15(1):54–64, Jan 1995.

[3] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proc. Intl. Conference on Measurement and Modelling of Computer Systems (ACM Sigmetrics 2000)*, 2000.

[4] Andrea C. Arpasci-Dusseau, David E. Culler, and ALan M. Mainwaring. Scheduling with implicit information in distributed systems. In *Proc. Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'98/PERFORMANCE'98)*, 1998.

[5] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proc.* 3rd *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1999.

[6] Ranieri Baraglia, Thomas Decker, Jorn Gehring, Domenico Laforenza, Friedhelm Ramme, Alexander Reinefeld, Thomas Romke, and Jens Simon. The mol project: An open extensible metacomputer. In *Proc. IEEE Heterogenous Computing Workshop*, pages 17–31, 1997.

[7] Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, and Charles V. Packer. Beowulf: A Parallel Workstation for Parallel Computing. In *Proceedings of the International Conference on Parallel Processing*, 1995.

[8] Rajkumar Buyaa, editor. *High Performance Cluster Computing (vol 1): Architectures and Systems*. Prentice Hall, 1999.

[9] Rajkumar Buyaa, editor. *High Performance Cluster Computing (vol 2): Programming and Applications*. Prentice Hall, 1999.

[10] K. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz. Directions in Active Networks. *IEEE Communications*, 1998.

[11] A. Chien, S. Pakin, M. Lauria, M. Buchanan, K. Hane, and L. Giannini. High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance. In *Proc.* 8th *SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[12] Steve Dawson, Marco Molteni, Livio Ricciulli, and Sonia Sui. User Guide to Anetd 1.6.3. SRI International, Sept. 2000.

[13] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12, 1996.

[14] Donald F. Ferguson, Christos Nikolau, Jakka Sairamesh, and Yechiam Yemini. Economic models for allocationg resources in computer systems. In Scott Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[15] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, 11(2):115–128, 1997.

[16] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[17] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM Parallel Virtual Machine – A User's Guide and Tutorial*. MIT Press, 1994.

[18] Peter Graham. A DSM Cluster Architecture Supporting Aggressive Computation in Active Networks. In *Intl. Workshop on Distributed Shared Memory (accepted)*, 2001.

[19] Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Paul F. Reynolds jr. Legion: The Next Logical Step Toward a Nationwide Virtual Computer. Technical Report 94-21, Dept. of Computer Science, University of Virginia, June 1994.

[20] W. Gropp, E. Lusk, and R. Thakur. *MPI-2 Advanced Features of the Message-Passing Interface*. MIT Press, 1999.

[21] Michael Hicks, Jonathan T. Moore, and D. Scott Alexander. PLANet: An Active Internetwork. In *IEEE Intl. Conf. On Computer Communications (INFOCOM'99)*, pages 1124–1133, 1999.

[22] Michael Hicks, Jonathan T. Moore, and D. Scott Alexander. PLANet: An Active Internetwork. In *IEEE Intl. Conf. On Computer Communications (INFOCOM'99)*, pages 1124–1133, 1999.

[23] L. Iftode, J. P. Singh, and K. Li. Scope Consistency: A Bridge Between Release Consistency and Entry Consistency. *Theory of Computing Systems*, 31(4):451–473, 1998.

[24] P. Keleher, A. Cox, S. Dwarkadas, and W. Zwaenepoel. Treadmarks: Distributed Memory on Standard Workstations and Operating Systems. In *Proceedings of the 1994 Usenix Winter Conference*, pages 115–131, 1994.

[25] M. Litzkow, M. Livny, and M.W. Mutka. Condor – A Hunter of Idle Workstations. In *Eighth International Conference on Distributed Computing Systems*, pages 104–111, 1988.

[26] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59:107–131, 1999.

[27] Multiview and Millipage Fine-Grain Sharing in Page-Based DSMs. Ayal itkovitz and assaf schuster. In *Proc.* 3rd *Intl. Conf. on Operating Systems Design and Implementation (OSDI'99)*, 1999.

[28] Rajesh Raman, Miron Levy, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proc.* 7th *Intl. Symp. on High Performance Distributed Computing (HPDC-7)*, 1998.

[29] J. Smith. Switchware: Accelerating Network Evolution. Technical Report MS-CIS-96-38, University of Pennsylvania, CIS Department, 1996.

[30] P. Sobalvarro, S. Pakin, W. E. Weihl, and A. A. Chien. Dynamic Coscheduling on Workstation Clusters. In *Proc. Intl. Parallel Processing Symposium*, 1998.

[31] Mark Swanson, Leigh Stoller, and John Carter. Making distributed shared memory simple, yet efficient. In *3rd International Workshop on High-Level Programming Models and Supportive Environments (HIPS '98)*, 1998.

[32] David L. Tennenhouse and David J. Wetherall. Towards an Active Network Architecture. *Computer Communication Review*, 26(2), April 1996.

[33] D. Wetherall, John Guttag, and David Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *1st Open Architectures and Network Programming Conference (OPENARCH'98)*, pages 117–129, 1998.

[34] David Wetherall. Active Network Vision and Reality: Lessons from a Capsule-Based System. In *17th ACM Symposium on Operating System Principles (SOSP99)*, pages 64–79, 1999.

[35] Rich Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proc.* 6th *Intl. Symp. on High Performance Distributed Computing (HPDC-6)*, 1997.