

Dynamic Cluster

Phuong N. Hoang, M.S.
Robert Chun, Ph.D.

Computer Science Department
San Jose State University
San Jose, California 95192
pnhoang@yahoo.com

ABSTRACT

This paper introduces a new architecture called the Dynamic Cluster, which is a cluster of computers connected together over a wired and wireless network. The nodes can use any network technology to connect to the cluster. The communication protocol of the Dynamic Cluster includes the dynamic detection of nodes entering and exiting the cluster. The Dynamic Cluster would dynamically redistribute task assignments in the cluster according to these dynamic changes in the cluster.

The Dynamic Cluster has the capability of distinguishing between time-constrained tasks and non-time-constrained tasks. Time-constrained tasks will have higher priority than non-time-constrained tasks and will be handled differently in load balancing and in “node-exit” events to guarantee that the time-constrained tasks will be processed on time.

The Dynamic Cluster’s manager nodes also have a real static IP address, allowing access from any machine with an Internet connection providing the Dynamic Cluster with a worldwide span. The sub-clusters of the Dynamic Cluster can be added or removed dynamically. They can also work independently.

Keywords

Parallel, Distributed, Architecture, Wireless, Cluster.

1. INTRODUCTION

Cluster Computing is becoming a very popular technology in parallel computing. In essence, a cluster computer is a collection of inexpensive computers networked together to work as a single computer under the management of a set of middleware. The main advantage of cluster computing is a much lower cost compared to other parallel computing technologies. Cluster computers are also much easier to maintain and update while remaining compatible with the programming semantics and syntax used by other types of parallel computers. As cluster computing becomes increasingly popular, the size of typical cluster computers has grown from a few dozen nodes to thousands of nodes with computing power in the order of TeraFLOPS.

However, because cluster computers are comprised of computers that must reside in the same physical location, the total computing power of the cluster is limited. There are other advanced parallel computing technologies such as grid computing that spans over the WAN network. Another example is SETI that can exploit the idle time of millions of computers around the world to perform useful work. Compared to IBM’s ASCI White, a massively parallel processing (MPP) supercomputer that yields 12 TeraFLOPS with a price tag of 110 million dollars, the SETI system can give 27 TeraFLOPS with a cost of approximately half a million dollars. The main difference between the two systems is that IBM’s ASCI White and similar systems use dedicated

computing units and specialized hardware, while SETI makes use of a much larger number of idle heterogeneous computing units around the world. Exploiting unused computing power has become a trend for parallel computing technologies.

The wireless technology boom provides the capability to extend this trend even further by allowing a greater number of computers to be connected to each other. This potentially adds a tremendous amount of computing power to the pool. However, the existing system architecture of parallel computing is not yet suitable to deal with the unreliable nature of a wireless connection. There is an existing pool of research for clusters of wireless computers, but it uses ad-hoc mode and creates separate networks of wireless computers instead of integrating them into an existing network infrastructure. Section 2 discusses the various architectures of existing cluster computing technologies in more detail. Section 3 proposes a new architecture, a Dynamic Cluster, that integrates wireless computers with wired cluster computers to exploit the idle time of laptops and desktops. The proposed architecture can be scaled gracefully for both LAN and WAN and can be integrated seamlessly into the existing network infrastructure. Section 4 describes some test results for the Dynamic Cluster.

2. RELATED WORK

2.1 Cluster Computers

By definition, a cluster computer is a “loosely coupled collection of standalone computing elements, usually in the same location and administrative domain” [14]. A cluster computer is much cheaper than a MPP supercomputer because it is comprised of commodity-off-the-shelf (COTS) computers networked together under the management of some middleware to function as a single computing resource. The computing units of a cluster computer can be individually replaced or upgraded by simply removing the old workstations and adding new ones. Similarly, the total computing power of a cluster computer can be augmented by simply adding more off-the-shelf computers to the network. The size of a cluster computer can also be very flexible, from a few dozen nodes to thousands of nodes. This flexibility makes cluster computers available to a wider range of users and budgets.

Cluster computers can have different architectures. At one extreme is the simple cluster architecture [14] in which users explicitly invoke programs to run on each node with minimal cluster middleware assistance. At the other extreme is the Single Image System architecture [17, 19] in which users can assume that the applications run on a single scalar computer; all distributed thread allocations and distributed memory allocations are transparent to the users. The cluster computer builder can choose any architecture in between the above two extremes. Custom cluster computer architectures may even be built for specific applications.

Cluster computers have many advantages. The first advantage is that cluster computers are usually much less expensive than their MPP counterparts because cluster computers can use off-the-shelf computers, commodity network hardware, and open-source Operating Systems and middleware. Another advantage is that the computing units of cluster computers are loosely coupled, so they are very easy to maintain, replace or upgrade. Therefore, cluster computers can quickly respond to changes in technology trends. Individual nodes can be swapped in or out with ease. Whenever a new chip technology is available to the PC market, the cluster can be upgraded by simply swapping old computers with new ones. Similarly, the network infrastructure of cluster computers can also be upgraded with ease. The next advantage of cluster computers is that they are highly scalable. Different from MPP supercomputers where the total power of the systems are fixed, a Cluster's computing power can change with ease by just adding more computers to the system. The last advantage of cluster computers is that they use the same software semantics and syntax for message passing as the commercial MPP counterparts.

However, cluster computers also have the following disadvantages. The first disadvantage is that all nodes in the cluster must be in the same physical location (e.g., same building) and the same administration domain. This limits the amount of computers that can participate in the cluster. Another disadvantage of a cluster computer is that all existing cluster software architectures rely on stable network connections between nodes. Although they are network latency tolerant, they are not network failure tolerant. Therefore, existing cluster architectures will not be suitable for integrating wireless nodes into the cluster.

2.2 Computational Grid

By definition, "A Computational Grid is a parallel and distributed system that enables sharing, selections and aggregations of geographically distributed resources including computing power, software, special devices and instruments, etc., across many different administration domains, depending on their availability, capacity, cost and user QoS requirements for solving large scale problems/applications" [3]. More and more organizations can afford to have their own supercomputers, and have found it beneficial to participate in a grid to share computing capacities. Grids allow organizations to contribute the idle time of their computer in exchange for the ability to utilize a large amount of computing resources when they need it. Although organizations have different incentives (money, benefit to human kind, ability of use a large aggregation of computing resources, etc.) and concerns (security, priority, network utilizations, etc.) when participating in a grid, these challenges can be answered by a grid architecture such as the Economic-Based Distributed System [3]. In this System, grid brokers and various helper services mediate interactions between buyers and sellers.

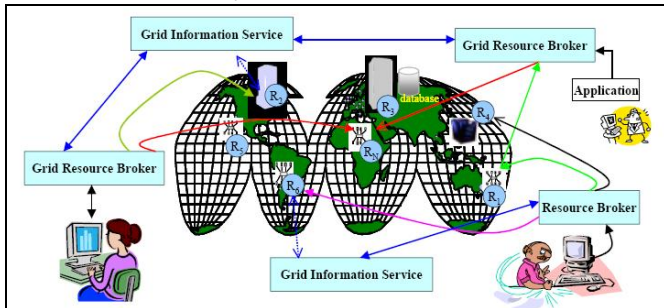


Figure 1: High Level View of Economic-Based Computational Grid

A Computational Grid employs the four layers of architecture as shown in figure 1:

- Grid Applications and Portal
- High Level Services and Tools (User-Level Middleware)
- Core Services (Core Middleware)
- Grid Fabric-Network (WAN network and storage)

When an organization wants to participate in the grid, it will register its resources with the Grid Market Directory as a Grid Service Provider (GSP). When users want to process a large application, they use tools in layer 2 to convert their application into smaller parallel processing units (called jobs) and submit them to the Grid Resource Broker (GRB) [4] with parameters such as budget and deadline constraints. The Grid Resource Broker will lookup the Grid Market Directory to find the best Service Providers to process the jobs. The GRB will also collect the job result and send it back to the users. A GridBank [2] maintains accounts for Grid Users and Grid Service Providers.

The Computational Grid has a well-defined architecture that satisfies most requirements of large-scale distributed and parallel systems. However, the architecture employs many layers of tools and services that would introduce significant overhead into the system. This would make the architecture better suited for a system of large-scale supercomputers than for an aggregation of individual personal computers.

2.3 SETI@Home

Search for ExtraTerrestrial Intelligence, or SETI [12], is a public project that uses the idle time of millions of personal computers around the world to analyze radio signals to detect extraterrestrial intelligence. Many terabytes of data will be collected by a radio telescope at the Arecibo Observatory in Puerto Rico and then sent to the SETI@Home lab at U.C. Berkeley. There, the data is divided into millions of 3.4KB work units and uploaded to the servers. Client machines (volunteered by their owners) then download these work units, analyze the data to search for some patterns and return any match as the result to the servers [1].

SETI@Home has a server-client architecture with the servers located at a well-known IP address on the Internet. The SETI server keeps track of millions of work units being sent to SETI clients around the world. Each work-unit is sent to multiple clients for redundancy. The result sent back from the clients is then crosschecked with the result from other clients processing the same work-unit for verification. The server also keeps track of all the clients and records the credit for each client participating in the project [1].

The SETI client program is a screen saver, which runs when the machine is idle. When the SETI program wakes up, it reads data from the hard disk and analyzes it. When it finishes analyzing the data, it will automatically connect to the server to send back the result. The SETI client can also be configured to ask the user's permission before connecting to the Internet or only transfer data when the user uses the Internet. If the machine becomes active, the SETI client will postpone the process and wake up again when the machine becomes idle [1].

The architecture of SETI@Home has many advantages. The first advantage is that it allows different types of computers from anywhere in the world to participate in the system without any modification to their system software. The users only need to download a screen saver that only executes when the machine is idle. The second advantage is it uses very little network bandwidth. The client software only uses the network connection

when it downloads the work-unit or uploads the result. Even during these communication periods, the client software uses very little bandwidth because of the small size of the work-unit (3.4KB). Because of these characteristics, almost all computers that have an Internet connection can participate in the SETI@Home project. The third advantage is that the work-units are redundantly sent to different computers and the results are crosschecked. When network communication fails or errors occur on the client side, the SETI system can still be guaranteed to provide an accurate result.

Although the SETI system has many advantages and can gather an enormous amount of computing power, it can only process jobs that have no time constraints. A SETI client program may download a work-unit but not complete it until after a few days or even weeks. Although the server can assign the job to different computers, the server itself has no control over the time it would be completed. If a time-constrained task is submitted to the system, the SETI server cannot guarantee having it done on time.

2.4 Wireless Cluster

A Wireless Cluster [9] is comprised of any number of machines, each of which has a wireless network card configured in ad-hoc mode and connected to other machines in a peer-to-peer network. Either the client software or the server software can run on any machine. The client machine can communicate directly with the server or it can use other client machines as relay agents to communicate with servers thereby extending the effective wireless range of the system. It is possible for one client machine to communicate with and receive jobs from multiple servers.

Initially, the client will start a routing discovery process by broadcasting its IP address in a message. Other clients will add its address to the message and rebroadcast it creating a forwarding path. When the server receives this message, it will add the client to the pool together with a routing path and send the message back to the client with that routing information. Intermediate nodes will relay the message.

After the routing discovery is completed, the client will regularly send load balancing information, including radio signal strength information, to the server in a beacon message. Because of the unstable nature of the wireless network connection and network topology, along with a communication bandwidth which can be proportional to radio signal strength, a wireless cluster uses signal strength as a major factor for load balancing decisions. For example, jobs requiring large communication bandwidths are sent to machines with the best signal strength. If the server does not receive any beacon message from a certain client for a long period of time, it will assume that the client is lost and it will remove the client machine from the pool. The client's routing information will also be removed from the routing table of the server.

When the server has a task to process, it will pick out one client according to the load-balancing algorithm and send the task to that client along with the routing path established during the routing discovery process. When the task is completed, the client sends the result back to the server. If the server assigns a task to a client and then receives a beacon message informing the server that the signal strength of the client is weakening (perhaps because it is moving out of range), the server will send the GetResult message to retrieve the temporary result immediately and reassign the task to a different client to finish the job. The server also reassigns the task to a different client if the server has not received any beacon from the client for a certain period, probably because that mobile client moved out of range [9].

The Wireless Cluster architecture has many advantages. It creates a way to tap into the computing power of an emerging group of mobile computers such as laptops, pocket PCs, etc. The Wireless Cluster provides a flexible architecture that allows wireless nodes to dynamically enter and leave the system. Since nodes in the network can relay each other's messages, the cluster can also be very scalable and extended well beyond the radio range of the wireless network card. Different from SETI, the Wireless Cluster can accept time-constrained jobs because the server can have certain control over a job's execution.

However, because the Wireless Cluster uses ad-hoc mode while existing wireless networks typically found in the work and home environment use infrastructure mode, it creates a separate network that cannot exploit the benefit of the existing network infrastructure. That is, the Wireless Cluster would only work in a LAN and cannot be extended to a WAN. This would limit the number of nodes participating in the cluster. Another disadvantage of the Wireless Cluster is that when the server leaves the network, the cluster disintegrates although other nodes are still available as clients. This makes the cluster very unstable.

3. DYNAMIC CLUSTER

The Dynamic Cluster is an attempt to combine the advantages of the architectures described in section 2, especially the advantages of SETI [1] and the Wireless Cluster [9], by supporting both time-constrained tasks and non-time-constrained tasks, and integrating both wired and wireless nodes, thereby extending the reach and scalability of the cluster to the Wide Area Network, allowing any machine on the Internet to participate.

The Dynamic Cluster is comprised of one or more sub-clusters. Sub-clusters can work independently or cooperate with others to distribute tasks. Each sub-cluster has a manager node and many worker nodes. Users can connect to any manager node to submit task requests and receive task results. When the manager node receives a task request, it will assign it to the best worker node for processing, factoring into account load balancing aspects.

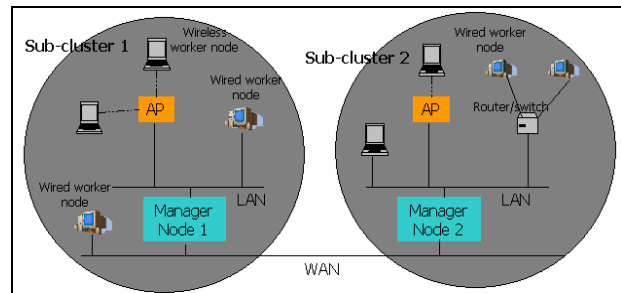


Figure 2: Dynamic Cluster Overview

3.1 Manager Node

Manager nodes connect via a wired connection to the WAN and have a real static IP address so worker nodes can connect to the manager node from anywhere in the world. Manager nodes communicate to each other in peer-to-peer fashion to share workloads. If a new manager node joins the cluster, it will communicate with the existing manager nodes to share load according to the load-balancing algorithm. Each manager node maintains the connections to all of its worker nodes and peer manager nodes and their resource information (including computing power, average network latency, current workload, number of worker nodes, etc.) to dynamically adjust the load balancing according to the status of the entire system.

Manager nodes are comprised of the three following components:

- The Resource Manager is responsible for managing connections to all worker nodes and peer manager nodes. The Resource Manager also does load balancing between worker nodes and load balancing with other manager nodes
- The Job Manager is responsible for receiving task requests from users, sending tasks to worker nodes, monitoring task executions and collecting results and sending the results to users
- The Display Manager is responsible for displaying the system status and graphical presentation of the cluster state

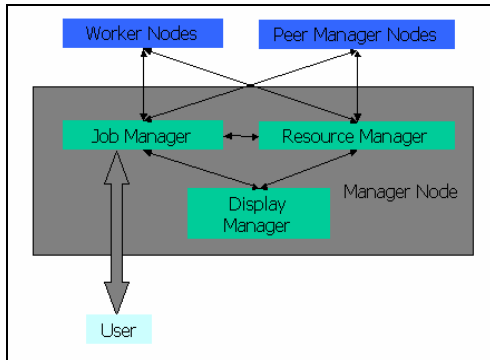


Figure 3: Manager Node Architecture

In addition to the three major components above, the manager node also uses a small module called `StatisticControl` to log all the task transactions in the manager nodes. The information being tracked by `StatisticControl` includes the time the task is received, assigned, reassigned, and completed, or forwarded or rejected which are later used in this report.

Internal Load balancing of each sub-cluster is done within the function `suggestHost` of the Resource Manager, which searches through the list of all hosts that are not overloaded to find the host that has the best combined (smallest) load value. For the sake of simplicity, the combined load value of each host is computed as follows:

$$\text{Value} = \text{cpuUsage} + \text{average Ping time} - \text{cpuSpeed} + \text{number of tasks currently processing} * 20$$

The smallest combined load value is equivalent to the combination of low CPU load, low network latency (average ping time), high computing power (measured in MFLOPS), and low number of currently running tasks. Also, if the task to be assigned is a high priority one, the combined load value is reduced by 20 if the host is "wired". This is done to favor the wired node over a wireless node to make sure that the high-priority tasks will be processed by a more reliable (wired) node.

The Resource Manager also has a list of peer manager nodes called an Active List. When the Resource Manager starts, it initializes the Active List of manager nodes by reading information from a configuration file (`config.xml`). The Resource Manager regularly sends probe messages to these manager nodes to exchange resource information such as number of worker nodes, current average load, total computing power, etc. The probe message also includes a time-stamp to check for freshness. The information in the probe message will be used in the external load balancing described in the next paragraph. If the Resource Manager cannot send to or receive any message from a manager node over a long period of time, it will consider communication

with that manager node lost and put that manager node on the Inactive List. When the Resource Manager receives a probe message from a peer manager node, it will check whether this manager node is in either the Active or Inactive List. If the peer manager node is in the Active List, the Resource Manager will update the resource information of the peer manager node. If the peer manager node is in the Inactive List, the Resource Manager will move it to the Active List and update its resource information. If the peer manager is not in either list, the Resource Manager will create a new item and insert it into the Active List. This feature enables the Dynamic Cluster to dynamically adjust to the entering and exiting of sub-clusters. The manager nodes in the Inactive List are probed less frequently than the Active List to detect any change in their connection status. If a manager node doesn't detect the presence of any other manager nodes, the sub-cluster will only work independently as a separate cluster and the manager node only sends task requests to its own worker nodes.

External (or inter-sub-cluster) load balancing will only take place if `suggestHost` cannot find a host because all hosts are overloaded or there is no worker node connected to the manager node. Then the Job Manager will call `suggestPeerServer` of the Resource Manager to search for a peer manager node to process the task. The Resource Manager has a list of peer manager nodes. In this implementation, the Resource Manager simply picks the first non-overloaded peer manager node in the list. When a peer manager node is picked, the Job Manager will forward the task request to that manager node. Because the list of peer manager nodes contains information such as the number of worker nodes, the average load, and the average network latency of each peer manager node, it is possible to do a more effective load balancing. With this forwarding mechanism, a manager node can use not only the resources of its worker node but also the available resources of worker nodes in all sub-clusters of the system.

3.2 Worker node

Worker nodes have an original list of known manager nodes. After booting, worker nodes will connect to one of the manager nodes.

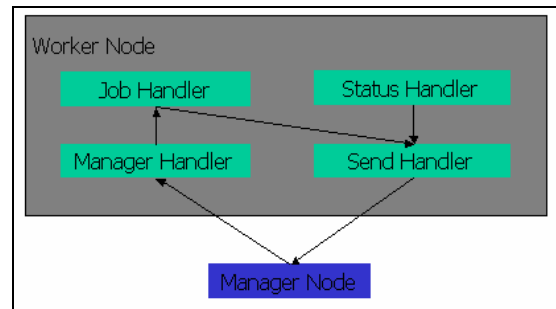


Figure 4: Worker Node Architecture

Worker nodes have the following components:

- Manager Handler which listens to the socket from the manager node to receive task requests
- Status Handler which will periodically query for local resource information (CPU load, connectivity, etc.) and send beacons to the manager to update a worker node's profile and maintain the connection to the manager node
- Job Handler which executes the tasks being assigned
- Send Handler which sends messages to the manager node

When the Manager Handler of a worker node receives a task request, it will add the task request to the jobQueue of the Job Manager. The Job Manager will loop through the jobQueue and remove task requests one-by-one. Each task request contains the:

- Name of the static method that returns an AppThread
- Parameters' name and values
- Name of the enclosure class of the method to be executed
- Task's Priority
- Java bytecode of the enclosure class and additional data files

With the data in the task request, the Job Manager will use a Java Reflection class to retrieve the enclosure class and invoke the method specified in the task request. When invoked, this method will create a thread, start it and return it to the caller. The Job Manager will call *join* to wait for this thread to complete. After the thread has completed, the Job Manager will post a task response message to the Send Handler to send back to the manager node. If the task request is a forwarded task request, the Job Handler will send two response messages. The first one is sent to the manager node to inform it of the completion of the task. The second one is sent directly to the original manager node that generated the task request. This way, the original manager node would receive the response at approximately the same time as the secondary manager node, and as a consequence, the user would not see any difference in response time between the forwarding and non-forwarding cases.

Under normal operation, the Status Manager will periodically ping the manager node to check the connection and measure the relative network latency. Ping is used instead of the signal strength from the wireless network card because the Dynamic Cluster includes not only wired but also wireless nodes. Besides, in the infrastructure mode, the signal strength only reflects the connectivity of connection from the node to the Access Point while the manager node is located behind the AP. In some cases, the worker node can be in a local network that connects to the Internet where the manager node is located. The Ping utility will be the best to measure the latency from source (worker node) to destination (manager node) regardless of how many hops there are between them. The CPU load is measured by utilizing a LoadInfo module developed by Myer [9]. This LoadInfo module uses Microsoft's Window Management Instrumentation (WMI) classes to retrieve system information such as CPU Usage, memory, disk space, etc. It is integrated into this project by the Java Native Interface. The ping average and the CPU load are packaged into a Beacon Message and sent to the manager node.

If the connection is lost, the StatusHandler and Manager Handler will be killed while the JobHandler continues processing the task requests in the jobQueue. SendHandler is also not killed, but all messages in the message queue are discarded except TaskResponse messages. SendHandler will attempt to send these messages as soon as the connection is reestablished. Because each message in the message queue contains the destination address, SendHandler can send messages to different manager nodes once the network connection is reestablished. However, because memory space is limited, old messages will be discarded if they are not delivered after a certain period of time.

If a worker node loses the connection to its manager node during task execution, it will continue to finish the task, and will send the result back to the manager node whenever the connection is reestablished. If the worker node was processing a high priority (time-constrained) task, the manager node will reassign all high

priority tasks to different worker nodes for processing. Low-priority tasks will not be reassigned by the manager and will wait for the connection to be reestablished. If over a long period of time, these low-priority tasks have still not yet been completed, the manager will reassign them to different worker nodes.

When a sub-cluster is overloaded, the manager node will choose a peer manager node and forward the task request to it. If the receiving manager node is also overloaded, it will reject the task request and the original manager node will choose another peer manager node. If all manager nodes reject the task request, the original manager node will send a reject message to the user. This is done to prevent system overloading.

When a worker node receives a forwarded task request, it will execute the task as normal, but will send the response message to both its manager node and the original manager node that generated the task request. As mentioned before, this enables the response message to reach both manager nodes at approximately the same time. The original manager node will then relay the response message to the user. In other words, when the worker node processes a forwarded task request, it behaves as if it belongs to both manager nodes. Internally, this feature enables the sub-clusters to virtually share their worker nodes; externally, this feature makes the forwarding transparent to the users.

4. RESULTS

The first experiment evaluated the performance of the Dynamic Cluster in comparison to a scalar computer. Several tests were executed using a fixed size matrix and various repetition factors, *r*. The amount of computing required for each task request is in the order of $r \cdot n^3$. With a fixed *n*, the amount of computing required for each task request will be directly proportional to *r*. So in this test, *r* is referred to as the computing intensity. The time measurement indicates the elapsed time from when a batch of task requests was submitted, to the time when all task requests in the batch were returned successfully. Each batch contained 2, 3, and 4 task requests with the same parameters (matrix size *n*, computing intensity *r*). The tests were first run locally on a scalar computer, and then compared with tests running on a 2 node cluster, 3 node cluster, and 4 node cluster.

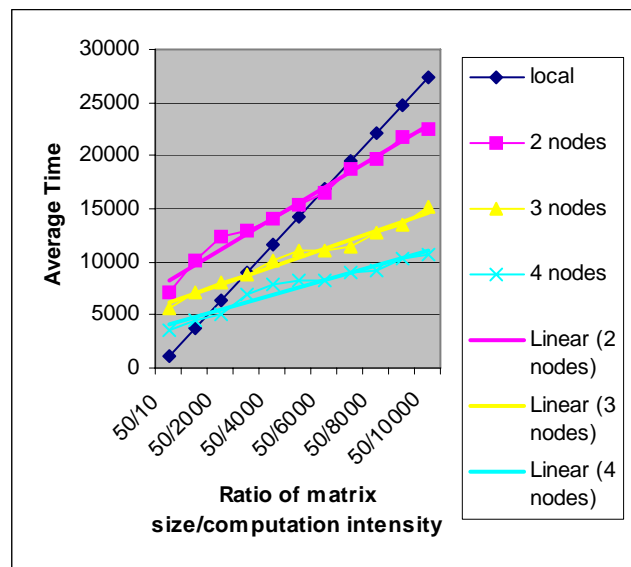


Figure 5: Comparison of average time to complete a batch of Task Requests on Dynamic Cluster and Single Scalar Computer

Figure 5 shows that the graph of the average computing time required to complete the batch of task requests on the local scalar computer is linearly increasing with the computing intensity. The Trendlines (computed by a least square fitting) of the graph of the average computing time on the Dynamic Cluster of 2 nodes, 3 nodes and 4 nodes also increased linearly with the computing intensity but they have a smaller slope than the average time on the scalar computer. As a consequence, the average computing time on the scalar computer started to exceed that of the Dynamic Cluster when the computing intensity increased beyond the point where the graph of the average time of the scalar computer cut through the trendlines of the average time of the Dynamic Cluster. This is because the Dynamic Cluster introduced an overhead for its management software and for transferring the task request over the network. So for I/O-bound tasks or small grain sized tasks, this overhead exceeded the benefit of having the task requests executing in parallel on the Dynamic Cluster. However, for compute-intensive tasks, this overhead was relatively small compared to the amount of time saved when the tasks were done in parallel on multiple nodes of the Dynamic Cluster and, as a consequence, the total time needed to process a batch of task requests on the Dynamic Cluster was smaller than that when running on the scalar computer. This means that with compute-intensive task requests, the Dynamic Cluster clearly outperforms the scalar computer. Therefore, the Dynamic Cluster is most effective for CPU-bound jobs.

The second experiment was done to determine whether the performance and total computing power of the Dynamic Cluster would increase when more computers were added into the cluster. The experiment was done in a fashion similar to the first one, but in this test, both the matrix size n and the computing intensity r were varied. Measurements were done using a Dynamic Cluster of 2 nodes. Then, the number of nodes was increased to 3 nodes, then 4 nodes.

The result in Figure 6 shows that with various ratios of matrix size to computation intensity, the average computing time required to complete the batch requests decreased as the number of nodes increased. This is because when more computers were added to the Dynamic Cluster, the more task requests could be done in parallel and, as a consequence, the total time required to complete the batch of task requests was reduced. Because this average time is inversely proportional to the performance of the Dynamic Cluster, it is safe to conclude that, for computing intensive (CPU-bound) jobs, the performance of the Dynamic Cluster increases as the number of cluster nodes increases.

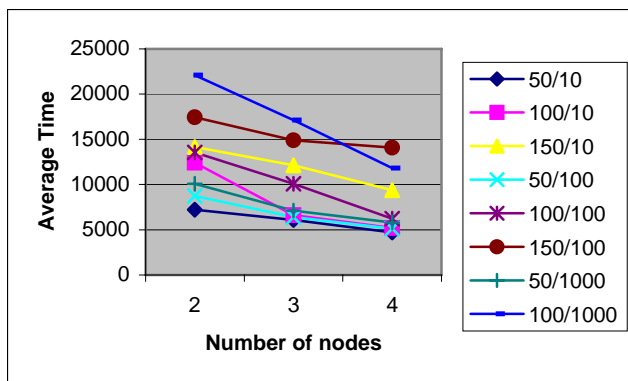


Figure 6: Average time to complete a batch of Task Requests on Dynamic Cluster of 2 nodes, 3 nodes and 4 nodes

The third experiment's goal was to measure the performance differences between processing task requests on the local sub-cluster vs. processing them on a peer sub-cluster. In the first run, the test driver was located in the same subnet with the manager node of sub-cluster 1. The test driver submitted multiple batches of task requests, each batch having a different matrix size/computation intensity ratio. The time measurement shows the time elapsed from when the test driver submitted the batch and the time when all tasks in the batch were completed. In the second run, sub-cluster 1 was connected with sub-cluster 2. Sub-cluster 2 only has a manager node and no worker nodes. The test driver was located on the same subnet as the manager of sub-cluster 2 and it submitted multiple batches of task requests to the manager node of sub-cluster 2. The time measurement was done in the same way as the first run.

The chart in figure 7 shows that for all ratios of matrix size to computing intensity, the average overhead of processing on a peer sub-cluster compared to processing on a local sub-cluster was less than 0.8% and, with a fixed matrix size, this overhead decreased as the computing intensity linearly increased. This overhead was reduced to approximately 0.3% when the computing intensity reached 5000.

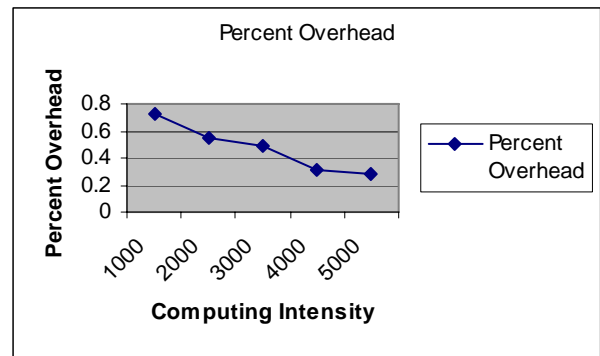


Figure 7: Average overhead of processing on a peer sub-cluster vs. processing on a local sub-cluster

This verified that the special forwarding, direct replying scheme used by the Dynamic Cluster described in section 3.2 has successfully produced a very small overhead. It also confirmed the result from experiment 1 that suggests the Dynamic Cluster is more appropriate for compute-intensive tasks.

5. CONCLUSION

The growing population of wireless computers generates a potentially enormous amount of computing power available for cluster computing technology. The Dynamic Cluster is a potential architecture that can incorporate wireless computers into a cluster. The Dynamic Cluster offers a blended solution that combines the benefits of SETI, Grid Computing, and the Wireless Cluster, and adds some unique features that make the cluster more available and easy to upgrade and update. The Dynamic Cluster detects the changes in network connection to each worker node or manager node in the cluster and dynamically adjusts to these changes. This enables the Dynamic Cluster to support both wired and wireless networks. The manager nodes of the Dynamic Cluster are located on the Internet, so any computer can become a part of the cluster as long as it has a connection to the Internet. This enables the Dynamic Cluster to be integrated seamlessly into an existing network infrastructure and also allows the Dynamic Cluster to have a worldwide span. In addition, the Dynamic

Cluster's load balancing algorithm and detection of network connection changes enable the cluster to support both time-constrained and non-time-constrained tasks.

One unique feature of the Dynamic Cluster is that it is comprised of multiple sub-clusters communicating with each other in a peer-to-peer fashion. Each sub-cluster has a manager node and multiple worker nodes. Sub-clusters can work independently or cooperate to share task requests. Because sub-clusters can work independently, the Dynamic Cluster is robust to network failures. This feature also makes the Dynamic Cluster more available to users because users can connect to any sub-cluster to have their task requests processed.

Another unique feature of the Dynamic Cluster is that if the worker node of sub-cluster 1 processes a task request forwarded from sub-cluster 2, it will send the task response to the manager nodes of both sub-clusters when it completes the task request. This reduces the total response time and makes it appear as if the worker node belongs to sub-cluster 2. In other words, this feature enables the virtual sharing of worker nodes among sub-clusters without adding extra complexity in software management.

While the Wireless Cluster technology [9] uses signal strength to detect the network connection changes, the Dynamic Cluster uses the ping utility instead to detect the network connection changes and to measure the relative network latency from the worker node to the manager node. This is an advantage because ping is universally available in all network technologies (LAN, Token ring, ATM, dial-up, wireless 802.11a/b/g etc.). Therefore, the underlying network infrastructure can change without affecting the Dynamic Cluster software.

6. FUTURE WORK

Several improvements could be incorporated into the Dynamic Cluster. Currently, the user can connect to any manager node but once the connection has been established, it has to be maintained until the process is completed. If the manager node crashes, the user has to restart communication from the beginning. This creates a single point of failure in the system. The project envisions a mechanism of sharing the user connections among manager nodes so that when one manager node crashes, others can take over its user connections and workloads.

In addition, the Dynamic Cluster currently rejects task requests if it is overloaded. The project can be improved by adding these task requests to a queue for later processing. The algorithm of load-balancing among sub-clusters can also be improved to choose a sub-cluster that has the best combined load value of total computing power, average network latency, and load.

Because the manager nodes are located on the Internet, the Dynamic Cluster is exposed to security threats. Therefore, adding security and access control would be good enhancements to the architecture. In addition to access control, account management for both users and worker nodes would make the project more appealing for public use. The account management system would manage user information such as username, password, and task statistics. It would also record the computing credits given to worker nodes or debits according to the number of task requests submitted and the cluster computing time used by the users. This account management may also require a database management system. Because the manager nodes are located on the Internet, enhancing the Dynamic Cluster with a web portal would enable users to log in, view their account information, submit new task requests, and view task results using an ordinary browser.

REFERENCES

- [1] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, Dan Werthimer, "SETI@Home: An experience in public-resource computing", *Comm. of the ACM*, Vol 45 No. 11, Nov 2002.
- [2] Alexander Barmouta, Rajkumar Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration", *International Parallel and Distributed Processing Symp. (IPDPS'03)*, pp. 245a, April 2003
- [3] Rajkumar Buyya, David Abramson, and Jonathan Giddy. "An Economy Driven Resource Management Architecture for Global Computational Power Grids".
- [4] Rajkumar Buyya, David Abramson, Jonathan Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid", *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region-Volume 1*, pp. 283, May 2000
- [5] Rajkumar Buyya, Kim Branson, Jon Giddy, David Abramson, "The Virtual Laboratory: A Toolset for Utilizing the World-Wide Grid to Design Drugs", *2nd IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGRID'02)*, pp. 278, May 2002
- [6] Jeffrey S. Chase, David E. Irwin, Laura E. Grit, Justin D. Moore, and Sara E. Sprenkle, "Dynamic Virtual Clusters in a Grid Site Manager", *12th IEEE Intl. Symposium on High Performance Distributed Computing (HPDC-12 '03)*, pp 90, June 2003
- [7] Thomas Fahringer, "JavaSymphony: A System for Development of Locality-Oriented Distributed and Parallel Java Applications", pp. 145, November 2000
- [8] Ting-Wei Hou, Fuh-Gwo. Chen, J.L. Lee, and Y.L. Cheng, "Distributed and Parallel Execution of Java Programs on a DSM System", *1st International Symposium on Cluster Computing and the Grid*, pp 555, May 2001
- [9] Chris Meyer. "Wireless Cluster Computing". 2004
- [10] "MPI: A Message Passing Interface Standard", *Message Passing Interface Forum (MPIF)*, May 1994.
- [11] Martin Placek, Rajkumar Buyya, "G-Monitor: A Web Portal for Monitoring and Steering Application Execution on Global Grids", *International Workshop on Challenges of Large Applications in Distributed Environments*, pp. 10, June 2003
- [12] [SETI@Home](http://setiathome.ssl.berkeley.edu/) Web site. Accessible on the Internet at World Wide Web URL <http://setiathome.ssl.berkeley.edu/>
- [13] M. Snir, S. Otto, S. Huss-Lederman, D. Walker and J. Dongarra. "MPI: The Complete Reference". MIT Press. 1998.
- [14] Thomas Sterling, J. Salmon, Donald Becker, D. Savarese. "How to Build a Beowulf: Guide to the Implementation and Application of PC Clusters". MIT Press. 1999.
- [15] Thomas Sterling, J. Salmon, Donald Becker, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, Charles Packer. "Beowulf: A Parallel Workstation For Scientific Computation".
- [16] Top 500 Supercomputer sites. Accessible on the Internet at World Wide Web URL <http://www.top500.org>
- [17] Bruce J. Walker, "Open Single System Image (openSSI) Linux Cluster Project". Retrieved from www.openssi.org on Dec 14th, 2005.
- [18] Rich Wolski, John Brevik, Chandra Krintz, Graziano Obertelli, Neil Spring, Alan Su, "Running EveryWare on the Computational Grid", *ACM/IEEE SC 1999 Conference (SC'99)*, pp. 6, November 1999
- [19] Wenzhang Zhu, Cho-Li Wang, Francis C. M. Lau, "JESSICA2: A Distributed Java Virtual Machine with Transparent Thread Migration Support", *IEEE Intl. Conference on Cluster Computing (CLUSTER'02)*, pp. 381, September 2002