

## **Creating A Virtual Computing Facility: Emulating Grid Services Reference Model<sup>[5]</sup>**

**Chris Clarke, Lu Marino, Ravi Pachigolla**  
School of Computer Science  
Pace University, 1 Martine Avenue, White Plains, NY.

### **Abstract**

Over the years, computers have become more powerful as their costs are declining. Most computers bought today are so powerful that having them in departments where they are used as dedicated workstations grossly under-utilizes their potential. This paper attempts to present a prototype for a high performance virtual computing facility created by co-opting idle and/or under-utilized computer workstations throughout acceptable departments to create a virtual computing pool. The idea is pretty simple and not new. It is gaining greater acceptance in academic institutions, which already possess enough potentially raw computing power to do distributive computing, without the need to buy high-end dedicated computer. What is really needed is a way to share that computing power throughout the system in a fair and consistent manner; with security concerns taken into account. We looked into a few possibilities with a feasibility test.

We looked at a dedicated cluster to see how much speed-up can be achieved through distributed computing then we went on to test the theory that Win32 computers, which make up the vast majority of workstation at our university, are powerful enough to be shared by the owner and sometimes a few other users who may need extra computing power to get their jobs done. With a set of priority policies that ensure that the owner's computing is always allocated the best share of the computing power – it is possible to accommodate outside jobs without interference. To demonstrate the feasibility of this, we choose a compute intensive molecular simulation package (NAMD) that comes with an internal loadleveler and ran tests in the student computer lab facility which is highly used by students during peak hours. Depending on the nature of the simulation, that is with respect to the number of atomic interactions that need to be calculated in real time – it was demonstrated that the more compute nodes attached to the simulator, the less intrusive the job was to the owner of any workstation in use. These tests are encouraging.

### **1. Introduction**

Cluster computing using networked commodity equipment has become an economic alternative for academic high performance computing facilities. Clusters aim to distribute compute intensive tasks over a set of backend nodes on a network in order to speed up the processing, and hence reducing the turnover time for a solution to a complex problem. As such, they have become powerful tools for tackling problems that were once beyond the scope of certain research programs. [10]

Building these clusters on an open source platform like Linux - a freely available incarnation of the Unix Operating System that targets Intel architecture, has made it easier for research, development, deployment and sharing of most major software packages developed in the academic community. Further, building Beowulf style Linux clusters have become easier with Open Source Cluster Application Resource (OSCAR)[11]. It incorporated diverse technologies to accomplish this task by creating a user-interface that abstracts away the details of the implementations making the task completely automated. When installed, OSCAR includes all the major open source software needed for parallel-distributed software development, the management tools for submitting parallel jobs, batch queuing, and policy control.

To use HPC clusters effectively – special messaging primitives are usually incorporated into the source code to enable task migration and coordination. The most popular open source packages are Message Passing Interface - MPI[16] and Parallel Virtual Machine[18] PVM. These libraries are freely available in most major programming languages.

Finally, in order to manage large scale deployment of clusters, special management software are needed to handle job submission queues and program execution details. These management software depend heavily on distributed management policies that dictate how particular domains or workstations will share their CPU resources. Most of these systems are aimed at taking the complexity out of managing users and the resources that are available to them. For instance, Grid Computing Environments[3] that handle all resource availability within the system, also control the communication subsystem, the way that resources are allocated, and determine the correct type of platforms available to set up the execution of a job delivered to the system. Grid Environments also present the user with a familiar set of services that they may use as a means of simplifying the process of launching a parallel application or simulation. [4] [5]

## **2. Objective**

Research and develop a prototype computer cluster network that is feasible and cost effective that fulfill these requirements:

- Harvest idle CPU cycles to run simulations from available computers on the network
- Easily deployed, configurable and managed by a network administrator
- Possess fine grain management for queuing user jobs

A very desirable research aim was to determine what packages existed that were cross platform i.e., interoperable within Unix/Linux and Windows NT cluster environment. This would greatly simplify configuration, management and deployment within the heterogeneous environment.

## **3. Initial Sub-Goal: A Beowulf Cluster**

We researched and implemented a Beowulf Cluster based on techniques found on the Internet as proof concept. The system was modestly constructed from 4 computer: three PII 266Mhz machines and a PI 166Mhz all running Linux 7.0. Each computer was equipped with a 10/100 Mbps LinkSys network interface card with the master node (the gateway) having an extra 100 Mbps Linksys card used to connect to the Internet. Additionally, the system was integrated using a Linksys 10/100 switch with each computer being assigned a private communication IP address 192.168.0.x and while the master node's second interfaced obtained an IP address from the university's DHCP server.

Once modifications were applied to the Linux system network infrastructure to achieve the requisite cluster-networking behavior [10] (i.e. enabling the direction of work flow to be away from the master node – while responses flowed towards the master node). The system was benchmarked to prove that we had done what we set out to do. We used a raytracing program that was adapted for parallel execution (PVMPOV[7]) to benchmark the system.

In general, all nodes should ideally have the same CPU to ensure that loads are equally distributed. Backend (compute) nodes with different speeds default to the speed of the slowest node depending on the type of processing done. Additionally, ideally, the coordinator or master node does not take part in calculations (it was used in this case since the problem was of the embarrassingly parallel category). This node may be slower than the compute nodes without adverse effects. A load balancing software may be employed to ensure that all workloads are evenly distributed based on feedback from the compute nodes:

### Test Runs

<b>BASELINE</b>	Run 1	Run 2	Run 3	Run 4	Average
PII 586/166MHz	203.60 s	204.52 s	202.26 s	203.15 s	<b>203.38</b>
PII 686/266	78.67 s	78.67 s	78.67 s	78.87 s	<b>78.72</b>

<b>Test 1 (2 Computer)</b>	Run 1	Run 2	Run 3	Run 4	Average
PII 266/266 MHz	70.67 s	69.30 s	69.40 s	70.10 s	<b>69.87</b>
PII 166/266 MHz	140.67 s	140.16 s	139.68 s	139.20 s	<b>139.91</b>

<b>Test 2 (3 Computer Net)</b>	Run 1	Run 2	Run 3	Run 4	Average
PII 266/266/266 MHz (N1/N2/N4)	62.39 s	63.42 s	62.69 s	62.27 s	<b>62.69</b>

<b>Test 3 (4 Computer Net)</b>	Run 1	Run 2	Run 3	Run 4	Average
PII 166/266/266/266 Mhz	31.84 s	34.72 s	32.97 s	31.81 s	<b>32.84</b>

Run time performance is dependent upon a number of factors besides CPU speed. The initial setup time of the job – which may involve transferring the program and its dependent files along with its runtime environment may detract significantly from the overall speed up if measures are not taken to minimize its impact. A toll on speed up may even occur if every compute nodes makes calls to the master node for large data files. In this case the overall bandwidth will become a bottleneck, which may also cause delays in start-up. Placing the programs in the file systems of the executing machine can mitigate this. While this might be a possibility, it does not necessarily scale well and is impractical way of managing an overly large cluster system.

A further point of interest would be the size of the portion of the data to be processed by each node. This parameter can be optimized to reduce bandwidth overhead on the network. Note also

that processing does not begin until the entire chunk of data arrives and certainly may lead to idle time for processor.

A final note is that using more compute nodes than is needed does not get the job done any faster. Load leveling was done manually by parameter arguments to PVMPOV. This is not the best way of handling this problem. But it give us a better understanding of the problem and what would be needed in a deployment over a network in which the software does not care about the environment in which it is placed in. In other words, not all parallel programs can be trusted to behave properly. It would be necessary to have a process monitor that is capable of affecting the NICE factor as well as the process priority in order to enforce workstation policies. This could complicate matters, as the slowest process will slow down the overall speed up factor of a parallel job. Since it is up to the person launching the program to know what may be the best requirements for their program – our side of the bargain should be to try and guarantee a certain CPU peak availability as a resource over the estimated time of the run. If this resource quality of service cannot be satisfied the job should be migrated or suspended.

#### **4. Phase II: The NAMD experiment**

NAMD[17] a scalable parallel object-oriented molecular dynamics program, developed at Beckman Institute, Illinois, as part of the Grand Challenge Application Group (GCAP) project. NAMD is one of the fastest molecular dynamics simulation programs that can scale to thousand processors. It takes an input file with the atomic structures of the system being simulated and distributes them to each node and runs simulation on each portion piping the trajectory data back to a file or to a simulation visualization client.

We installed a memory resident client called Charm on all machines in the computer lab facility to facilitate communication between the Win32 machines in a Parallel Virtual Machine fashion. When the client version of Charm is started with NAMD as its argument – it reads the node lists file containing the names of the computers on the network with resident clients running. This creates a virtual computing pool within which NAMD will communicate and coordinate its activities. NAMD receives the number of processes that it should spawn as a command argument. It then spawns these processes on the available workstations that is made available from Charm. The NAMD directory was cross-mounted across the network so that each process loads from the same environment on all nodes in the pool and can read its input portion from the protein definition input file.

Once the simulation is started, we used VMD[19] to attach to the originating Molecular Dynamics simulator client (NAMD) to view the real-time simulation from any workstation connected to the network. NAMD creates a .dcd trajectory file that is saved in the home directory so that the simulation can be re-run at a future time. VMD currently only allows one observer. The source code is freely available and can be modified for multi-viewership.

NAMD does periodic load balancing across the workstation pool in an attempt to distribute the workload evenly so computer owners are not affected by its presence on their workstation. It does statistics on the CPU usage, amount of memory available, and processes usage from all machines to determine the best way to recalibrate the NAMD process priority on the remote machine(s).

Conclusion:

NAMD successfully demonstrated that with the continued increase power of CPU in Intel workstations, it is now possible to clandestinely steal CPU cycles on machines within a virtual pool without overtly affecting the User logged into the machine. It is quite possible to migrate processes to a CPU that is lightly loaded, without becoming intrusive to the user these days. NAMD was able to monitor its behavior of its remote processes over the network. Computing the atomic trajectories is compute intensive – so that the more NAMD clients running on a pool of workstations that is with respect to the number of workstations available within reduced the overall effect of its presence on any single workstation. Also noticeable was the dramatic reduction of average CPU usage on the originating machine even though the amount of coordination activity has to increase to facilitate more remote clients as the processor pool increased. NAMD is exceptional in that its overall performance within the pool was load leveled to ensure fair usage of the individual CPUs within the pool. That is, the more processors being utilized within the pool, the less work individual CPU have to do thus ensuring that other jobs submitted to the pool or to a workstation within the pool will be equally well attended by the CPU.

## **5. Condor – A Case Study For A Virtual Computing Facility**

Condor[12] is considered a High Throughput Computing software system whose aim is to maximize the utilization of all computing resources (workstations that communicate over a network) through distributed ownership. The Condor System consists of a pool of computers. Each node in the pool can act as compute node where jobs can be executed, as compute node where jobs can be submitted and executed, or as an exclusive submit node. A Central Pool Manager exists on a node and is responsible for managing resource availability and process allocation using policies dictated by the administrators and workstation owners in the pool. Condor uses a ClassAds mechanism that acts as a match-maker for resource owners and resource consumers within the pool thus ensuring that requirements for a job is enforced.

Condor possesses the following features:

- Condor makes available resources more efficient by putting idle machines to work. It expands the resources available to users, by functioning well in an environment of distributed ownership.
- Condor has a set of tools to submit, run and manage queues of jobs running on a cluster(s) of NT machines. It also possesses a set of tools that make the job of controlling the queuing and batch submission manageable.
- It has customizable job policies using a ClassAds mechanism. Machine ClassAds contain information on the current load average, RAM and virtual memory size, integer and floating-point performance, keyboard/ mouse idle time, etc. JobClassAds contain a full complement of information, including system dependent entries such as dynamic updates of the job's image size and CPU usage. Jobs want to find machines upon which they can execute. A job will require a specific platform on which to execute. Machines have specific resources available, such as the CPU speed and the amount of available memory. A separate ClassAd is produced for each job and

machine, listing all attributes. Condor acts as a matchmaker between the jobs and the machines by pairing the ClassAd of a job with the ClassAd of a machine.

- Condor possesses tools necessary for central management and several security mechanisms. All jobs are run in the background so that any windows created by a job cannot be seen by anyone using the machine.
- Condor supports SMP machines. Additionally it can run jobs at a lower operating system priority level. Jobs can be suspended, soft-killed, or hard-killed automatically based on policy expressions. For example Condor NT can automatically suspend a job whenever keyboard/mouse or non-Condor created CPU activity is detected, and continue the job after the machine has been idle for a specified amount of time.
- Condor NT correctly manages jobs which create multiple processes. Additionally, besides interactive tools, users and administrators can receive information from Condor by email (standard SMTP) and/or by log files.
- Unix and Windows NT machines running Condor will happily co-exist in the same Condor pool without any problems. The only restriction is jobs submitted on Windows NT must run on Windows NT, and vice versa. This behavior is by default specification in the job submission *requirements* field in the submission file that is sent via the *job\_submit* command.
- Condor pools are scalable; as quantity of computational resources available to the users grows to hundreds of machines. The ClassAds mechanism can be made as flexible as needs to be. A job may also prefer to execute on a machine with better floating point facilities, or it may prefer to execute on a specific set of machines. These preferences are also expressed in the ClassAd. Further, a machine owner has great control over which jobs are executed under what circumstances on the machine. The owner writes a configuration file that specifies both requirements and preferences for the jobs. The owner may allow jobs to execute when the machine is idle (identified by low load and no keyboard activity), or allow jobs only on Tuesday evenings. There may be a requirement that only jobs from a specific group of users may execute. Alternatively, any of these may be expressed as a preference, for example where the machine prefers the jobs of a select group, but will accept the jobs of others if there are no jobs from the select group. In this way, machine owners have extensive control over their machine. And, with this control, more machine owners are happy to participate by joining a Condor pool.

Based on the above feature list – we believe that Condor embodies the requirements to create a virtual computing pool facility. We installed the Condor version for Win32 on a NT4.0 (200Mhz w/64M RAM) and an XP box (1.8Ghz w/512M RAM) and tested for job submission, and batch queuing. These tests were successful, however we could not get a multi process program that ran on Win32 to test before this writing. We are optimistic that Condor will be capable of handling the details.

## **6. General Conclusion:**

In general, the scenario that software programs should dynamically alter its behavior based on workstations availability within a dynamic processor pool would be placing a burden upon the programmer. What would be a better model is that the program should be capable of requesting resources that fulfill certain requirements and the Environment in which it is running in should be capable of locating, allocating and guaranteeing these services.

For instance, our complex setup to run our simulation experiment rather than going to each workstation to launch the memory resident Charm daemon, then running the Charm client which launches the NAMD with an input file requires that the researcher know details about the virtual pool. A job submitted to Condor theoretically would first determine the necessary compute requirements from the submitted request file. It would then check through its ClassAds to determine if the resources are available, and whether the quality of service could be guaranteed to some degree of probability. Once the virtual pool is established – it would make this list available to NAMD. It would then launch the Charm client that takes NAMD as its input. When the simulation is completed Condor would ensure that – the Charm daemons are removed from all workstations' memory. With sufficient policy setting it is also possible to ensure that NAMD does not over step its requested CPU usage parameters. This is a cleaner solution than having to manually launch or placing the Charm resident program into batch files to be loaded at workstation start up.

Condor is a perfect choice for creating a virtual computing facility, since the submit nodes can be in any department – allowing the creation of a virtual department. Condor enables the creation of such a virtual department with instituted user rights and policies through the ClassAds mechanism as well as the distributed local policies.

## **7. Recommendations and Future Achievable Goals**

The highlight of this paper is that users that comprise a virtual department require services that may be readily available on a system of networked workstations. These resources are governed by varying policies in different domains and on individual workstations, which complicates rather than simplifies this very simple concept. However, even with distributed policies it is conceivable that these workstations can be easily incorporated into a virtual computing pool with the ClassAd mechanism.

We have looked at working cases of this system as featured on the Internet. We found the resource services model to be the backbone of Grid Computing. The Globus Toolkit[11] using the Grid Services Reference Model [2] demonstrates how this infrastructure that can be leveraged into an academic facility that would enable this form of virtual computing department to be formed on the fly, at least from a theoretical perspective.

PUNCH[6], Purdue University Network Computing Hubs, is an actual example of the services perspective of Grid Computing. This initiative presents a web-based interface as a front-end to their Cluster resources. The developers consider that both Portable Batch System and Condor are too low level for the casual user/researcher who only needs to get their problem solved instead of bothering with the details of how to get the job submitted and executed. Their web interface take away all these low level details and acts as a single point of entry into their system – taking away all the complexity of needing to know either of the management systems.

Condor takes advantage of computing resources that would otherwise be idle and puts them to good use. Condor streamlines the scientist's tasks by allowing the submission of many jobs at the same time. In this way, tremendous amounts of computation can be done with very little

intervention from the user. Condor provides other important features to its users. Source code does not have to be modified in any way to take advantage of these benefits. Code that can be re-linked with the Condor libraries gains two further abilities: the jobs can produce checkpoints and they can perform remote system calls.

A checkpoint is the complete set of information that comprises a program's state. Given a checkpoint, a program can use the checkpoint to resume execution. For long-running computations, the ability to produce and use checkpoints can save days, or even weeks of accumulated computation time. If a machine crashes, or must be rebooted for an administrative task, a checkpoint preserves computation already completed. Condor makes checkpoints of jobs, doing so periodically, or when the machine on which a job is executing will shortly become unavailable. In this way, the job can be continued on another machine (of the same platform); this is known as process migration.

Condor-G (the Grid Services version) – takes this further by enforcing policies across a grid, enabling users to be able to run their jobs in virtual pools within any organization that is a part of the grid. We expect that the university will adopt grid computing for the many benefits that it will offer in future.

## References

- [1] T Bemmer. RWTH Scalable Computing Multi-platform MPICH (MP-MPICH) 2/28/02  
<http://www.lfbs.rwth-aachen.de/users/joachim/download>
- [2] F. Berman, A. Chin et. al. *The Grads Project: Software Support For High Level Grid Application Development*. 2001.
- [3] R. Figueiredo, N. Kapadai, J. Fortes. *The PUNCH Virtual File System*. IEEE Intl. Symposium on HPDC, Aug. 2001.
- [4] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.
- [5] I. Foster, C. Kessleman and S. Tuecke. *The Anatomy Of The Grid: Enabling Scalable Virtual Organizations* Intl. Journal of Supercomputer Applications, 2001.
- [6] W. Gropp, E Lusk, A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 1994.
- [7] B. Kline, A. Dilger. *PVMPOV How To*. Sept. 2001.
- [8] J Marinho, J. G. Silva. *WMPI: Message Passing Interface for Win32 Clusters*. 1998.
- [9] J. Phillips, J. Stone, T. Skirvin. *Designing A Cluster for a Small Research Group*. Low Cost Linux Clusters for Biomolecular simulations using NAMD. (1998).  
<http://www.ks.uiuc.edu>
- [10] Thomas Sterling. *Beowulf Cluster Computing with Linux*. MIT Press 2002. ISBN 0-262069274-0

- [11] Cluster In A Box: *OSCAR 1.2.1*. Feb 2002. OSCAR version 1.2.1 is a snapshot of the best known methods for building, programming, and using clusters. It consists of a fully integrated and easy to install software bundle designed for high performance cluster computing. Everything needed to install, build, maintain, and use a modest sized Linux cluster is included in the suite, making it unnecessary to download or even install any individual software packages on your cluster. <http://oscar.sourceforge.net>
- [12] Condor: the goal of the Condor Project is to develop, implement, deploy, and evaluate mechanisms and policies that support [High Throughput Computing \(HTC\)](#) on large collections of distributively owned computing resources. <http://www.cs.wisc.edu/condor/>
- [13] Globus Toolkit: The Globus Project is developing fundamental technologies needed to build [computational grids](#). Grids are persistent environments that enable software applications to integrate instruments, displays, computational and information resources that are managed by diverse organizations in widespread locations. <http://www.globus.org>
- [12] Grid In A Box  
<http://www.ncsa.uiuc.edu/UserInfo/Alliance/TechnologyGrid/GiB/>
- [16] MPI for win32) MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users. <http://www-unix.mcs.anl.gov/mpi/>
- [17] NAMD is a parallel, object-oriented molecular dynamics code designed for high-performance simulation of large biomolecular systems. NAMD [scales](#) to hundreds of processors on high-end parallel platforms and tens of processors on commodity [clusters](#) using switched fast ethernet. <http://www.ks.uiuc.edu/Research/namd/>
- [18] PVM (Parallel Virtual Machine) is a software package that permits a heterogeneous collection of Unix and/or NT computers hooked together by a network to be used as a single large parallel computer. [http://www.csm.ornl.gov/pvm/pvm\\_home.html](http://www.csm.ornl.gov/pvm/pvm_home.html)
- [19] VMD is a molecular visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics and built-in scripting. VMD supports computers running MacOS-X, Unix, or Windows, is distributed free of charge, and includes source code. <http://www.ks.uiuc.edu/Research/vmd/>