

Ideal Network Enabled Server Environment

Pushpinder Kaur Chouhan, Adarsh Patil and John P. Morrison

Department of Computer Science
University College Cork
Cork, Ireland.

Abstract Network Enabled Server environments developed quickly in the mid 1990s. Different approaches adopting diverse technologies have emerged and these environments have continued to evolve. In this paper we have envisioned the characteristics of an ideal NES environment for the perspective of usability and efficiency. It is hoped that this information, synthesized from Surveying many Network Enabled Server Environments, will facilitate developers in improving these environments. The perspective of administrators, application vendors, developers, owners and users on popular Network Enabled Server Environments are presented.

Keywords: Network Enabled Servers, Distributed Computing, Middleware

1 Introduction

Eventhough the processing power of computers is steadily increasing there are, and possibly will always be, a large class of problems that can only be solved using collections of machines operating in concert. Thus, Network Enabled Server (NES) environments are being developed to provide end users (scientists, biologists, mathematicians, astro-physicists, etc.) with access to computational facilities via machines with the potential to solve their applications.

NES environments are characterized by a client-server computation model distributed over a wide area network and applications which typically require much more CPU cycles and data storage than is available locally. Theoretically all the NES environments have the same goal, but these environments are many and differ according to their underlying computing models.

This paper outlines an ideal NES environment in which important execution characteristics are easily exploitable by each stakeholder. In doing this, existing NES environments, including DIET,

NeOS, NetSolve, Ninf, Nimrod, PUNCH, and WebCom, are surveyed and their stakeholder relationship are examined. By collecting this information into one place stakeholders are aided in choosing an NES environment to best suit their needs.

This paper is organized as follows: Section 2 describes the basic features of an ideal NES environment with respect to the stakeholders' perspectives. Section 3 presents the characteristics of an ideal NES environment. Section 4 describes some of the existing NES environments with respect to the demands presented in Section 2. Section 5 mentions some systems that are similar to NES environments in providing services but which differ in techniques. Section 6 draws some conclusions.

2 NES environment with respect to different perspectives

An ideal NES environment is one that can fulfil all the requirements of its stakeholders. As the number of resources increases in an NES environment, the complexity in its use and development is effected and so are the stakeholders of the NES environment. Figure 1 shows the relation between the number of resources and the utilization complexity of stakeholders of NES environments.

The main characteristics of an ideal NES environment include stability, ease of configuration, secure connectivity, ease of accessibility, guarranty of success, high throughput, optimal resource allocation, efficiency of cycle usage, efficiency of memory usage, low latency, fast and secure data transfer and self repairing. In this section, an ideal NES environment is specified from the perspectives of administrators, application vendors, developers, owners and users.

2.1 Administrator

NES environment administrators are responsible for maintaining stable NES environments required by their users. Security and authentication are the main concerns for the administrator. Thus, ideal NES environments with respect to administrators, are which provide proper tools for managing the user and underlying hardware profiles. Tools should be autonomic in nature exposing self managing, self healing and self co-ordinating characteristics so as to keep the software and hardware stability intact. This autonomic behaviour will lessen the burden of respective domain administrative personnel. It will also help administrators to track and solve problems quickly and efficiently.

2.2 Application Vendors

NES environments execute user applications. Users can be novices with respect to programming languages and/or the tools that are required to convert their applications into the acceptable formats. Thus, the work of application conversion falls to application vendors. To facilitate this task, NES systems should be configured to reflect users requirements rather than from hardware/systems perspective. The utilization of an NES environment should be independent of application development. Moreover, an NES application format should be independent of its underlying resources. For the application vendor an ideal NES environment should accept applications written in any language and should be compatible for all platforms.

2.3 Developers

These are the personnel responsible for the core development of NES systems. An ideal NES environment for the developer is one that is developed in a modularized and pluggable manner. These modular and pluggable characteristics allows existing and novel developers to extend and add new functionality to NES environments in an easy way.

2.4 Owners

The owners are those who own the resources hosted in an NES environment. The main concern of owners is user and resource management to provide good quality of service. They balance this with the cost of ownership in solving hardware and software issues of configuration and installation. An NES environment should properly implement the security and accounting methods, which should be easy

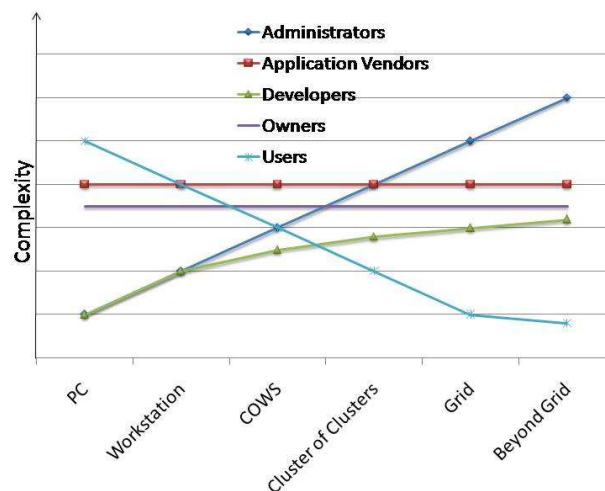


Figure 1: Complexity of various user types with scalable resources.

for owners to access, control and update. Owners are concerned with the productivity of their resources. An ideal NES environment for an owner is one that maximize return on investment and reduces management cost.

2.5 User

An NES environment is used by different kind of end users. End users typically come from many application domains. Not all are computer experts and require varying degrees of support. The NES installation should provide single sign-on and secure connection to all the resources in the NES environment from their machine; keeping the burden of multiple logins away from the user. The NES environment should provide a discrete user space for securely staging data and results. There should be one interactive window to the entire NES environment providing services with respect to job/data management, job/data monitoring, job/data query, job/data recovery, resource discovery, configuration and resource reservation.

For the end users an ideal NES environment is one which is easy and straight forward to install and use. An ideal NES environment should be stable and autonomic with respect to the dynamic nature and potential problems of underlying resources so that users are unaware of any changes that occurs.

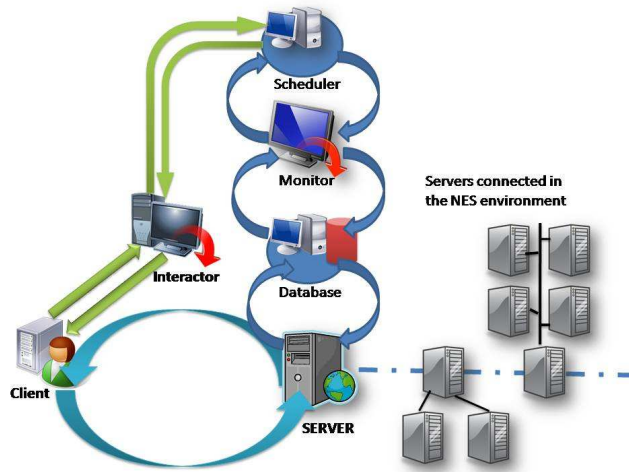


Figure 2: An Ideal Network Enabled Server environments: *Clients* provide user interface to submit requests for execution. *Servers* receive requests from clients and execute libraries or applications on their behalf. The *Scheduler* selects a potential server from a list of servers maintained in the database.

3 Ideal NES environment

An ideal NES environment, which attempts to maximize the advantages to each stakeholder is difficult to achieve since individual stakeholder's requirements may directly conflict. For example, an owner may wish to minimize cost of ownership whereas a user may wish to maximize resource heterogeneity. Trade-offs are therefore inevitable and these may be motivated by local considerations that change over time. Nevertheless, there are improvements that can be made to viewed appropriately advantage all stakeholders and disadvantage none. These are considered to form a highly desirable core and are describe here.

3.1 Components

To maximize flexibility in constructing an NES environment, it is useful to consider each functional unit as a separate component, eventhough this to not the traditional view. The following functionality has traditionally been considered as core: a client, a server, a database, a monitor and a scheduler. Experiencing each of these functional units as components which can be optimally combined and deployed is the first step in defining an ideal NES environment.

Another addition, which aims to assist the proper launching of user applications is the "Inter-

actor Component". It is proposed that this component be able to dynamically accept user input and to be able to verify the correctness of user parameter types. This component may be deployed with the client user-interface or separately if required. Figure 2 shows a deployment of the specified components in which a separate machine to used to host each component.

3.2 Architecture

An ideal NES environment should support both flat and hierarchical architectural models. In a flat model all users submit requests to a single client, which is directly connected to all available servers. The client does the work of the scheduler in selecting an appropriate server for submitted requests. In a hierarchical model, a top-down tree of clients are arranged in such a manner that scheduling requests can be shared among the available clients.

The architecture should be dynamically changeable and client should be able to perform the functionality of servers and vise-versa. In short, an ideal architecture should support both a client/server architecture and a Peer-to-Peer architecture to maintain stability.

3.3 Task Execution

An ideal NES environment should be properly initialized. First the interactor and then a component that act as a scheduler should be initialized followed by the launch of computational components. Finally, the component is launched with which the user can submit requests.

The working of an ideal NES environment is described below. *Monitors* should frequently monitor the status of the available machines (network, latency, CPU load, etc.) and register the information to the *Database*. The *Interactor* checks the correctness of the submitted application with respect to the user submitted application requirements. When submitted an application is verified, the interactor then asks the scheduler to provide a suitable computational server for the client's request. *Clients* provide an interface for users to submit their requests with the help of client APIs (e.g. C, Java, MATLAB, Fortran, Web, e-mail, etc.) or tools constructed using these client APIs. The *scheduler* queries the database and selects a suitable computing resources based on certain algorithms and returns the selection to the client through the interactor. Clients remotely invoke these libraries (or applications) on the selected

server. The client sends the reference of the selected server to the user so as to directly submit their application and data to the server for execution. The *server* does the computation and returns the resulted data to the client.

3.4 Scheduling

The scheduling of submitted applications has to be done on two levels: firstly, a client selects an appropriate server according to submitted request; and secondly, the scheduling of application on the servers queue. The client should implement the scheduling algorithm that can choose the server, which meets the application deadline, and user defined plug-in schedulers. The position of an application in the server queue should be such that the specified execution deadline is met.

3.5 Data Management

Once the selection of the server is made the user should directly send input data to it for execution. The server executes the application and generate the result data, which can be handled in one of 2 ways: if the data can be used by another application, it is stored on the server and transferred when and where required. It is sent back to the user either directly or indirectly through the scheduler.

3.6 Fault Tolerance

An NES environment should have two level of fault tolerance: application level and component level. At the application level, checkpointing techniques can be used. In checkpointing, snapshots of the execution are stored on another machine so that execution can be restarted from the last checkpoint. Data checkpointed at an earlier time can be removed after a certain amount of time that can be fixed by the user or the administrator.

At the component level, ancestor listing is used as a means for recovering from faults. Each component keeps a list of its nearest ancestor, so that if a parent component fails, it tries to reconnect to the nearest living ancestor.

3.7 Security

Applications are run anonymously within an NES environment. Security is a must for client code and for the owner of the machine on which the code is running. Security is important for many reasons: resources typically belong to multiple owners, users

worry about having private data exposed on a public network, and there is a real need to prevent malicious use of valuable shared resources.

3.8 Miscellaneous

The above mentioned characteristics are most important in an ideal NES environment. Here other characteristics that can help users to efficiently use an NES environment are mentioned. To facilitate the user, application submission should be easy. An ideal NES environment should not take a specific input type, it should have different automatically translating modules that can convert any type of application to the required type that an NES environment can process.

An NES environment contains a heterogeneous set of machines that run on different operating system and hardware, so the component coding techniques should be compatible.

An NES environment should use deployment techniques to choose the component set, describing number of machines, role played by these machines and their organization according to submitted jobs. This information should be used to deploy the NES platform automatically [9]. To check the status of connected NES components, any visualization tool should be useable. Visualization keeps users in contact with their executing code and promotes a feeling of transparency. It can also be important in the implementation of computation steering techniques and in the development of fault tolerant algorithms.

4 Existing NES environments

This section outlines some of existing NES environments. The sequence of these NES environments in the article is in an alphabetic order DIET, NeOS, NetSolve, Nimrod, Ninf, PUNCH, and WebCom.

4.1 DIET

Distributed Interactive Engineering Toolbox [6] is developed at École Normale Supérieure de Lyon. DIET is freely available for download under respective licence for all flavors of UNIX and Windows environment. DIET is user friendly system. It provides deployment and visualization tool. These tools promote easy deployment and monitoring of the system. In DIET users can use a plug-in scheduler [2] for server selection. For users DIET provides fault tolerance by using Chandra and Toueg and Aguilera failure detector [8]. DIET is not very specific for application vendor and administrator as

it uses third party middleware (CORBA) for communication. For data management, DIET gives two tools which can be easily operated by administrators and application vendors. DIET does not have any inbuilt security. DIET uses VPN (Virtual Private Networks) and CORBA (Common Object Request Broker Architecture) for secure connections between its client and server applications. This might be a problem for administrators and application vendors.

4.2 NeOS

Network-Enabled Optimization Server [4] is developed at the Argonne National Laboratory. It is available for Win 9x/XP and all flavors of Unix. NeOS implementation and interfaces are written in Tcl/Tk, Java and Kestrel. As it is a static environment to solve optimization problem, NeOS servers are initialized by the NeOS administrator, users cannot initialize the NeOS server, they can only submit their job for optimization to NeOS server. NeOS communicates with the user through Internet communication using TCP/IP. Security in NeOS is not inbuilt. VPN security is used by NeOS for secure connections between the application sent and the server.

4.3 NetSolve

Network-enabled Solver-based [7] system is developed at University of Tennessee, Knoxville. It is available for all popular variants of the UNIX operating system, and part of the system are available for the Microsoft windows platforms. The NetSolve system is implemented using the C programming language, with the exception of the thin upper layers of the client API that serve as environment specific interfaces to the NetSolve system. NetSolve provide some beneficial tools for user like visualization tool to display the status of the resources interacting with the NetSolve system. However, the selection of server is not very effective as it uses a theoretical model [3] to estimate the performance, given the raw performance and the CPU load. Owners of server nodes are better than owners of agents because if agent is crashed, NetSolve system can not be used, where as if NetSolve server got crashed, then request can be resubmitted to other server. For data management NetSolve provides two approaches, which are helpful for application vendors and users for easy data access and management. For application vendors and administrators it might be difficult to maintain the systems

performance as it uses TCP/IPv4 sockets and a NetSolve-specific application layer protocol to communicate with each other. Even security is enabled via Kerberos support.

4.4 Nimrod

Nimrod [1] is developed at School of Computer Science and Software Engineering Monash University. It is implemented in the C language. Nimrod-G provides a persistent and programmable task-farming engine that enables 'plugging' of user-defined schedulers and customized applications in place of default components. It takes the experiment plan as input, described by declarative parametric modeling language (the plan can also be created using the Cluster GUI) and manages the experiment under the direction of schedule advisor. The experiment has to be restarted if the node running Nimrod goes down. Parametric Engine crash leads to unavailability of Nimrod environment, thus for administrators and users it is very risky to access Nimrod when system is not stable. Nimrod database is containing routing information that is constructed, accessed, and acted upon by the routing functions.

4.5 Ninf

Ninf [13] is a collaboration product of AIST, University of Tsukuba, Tokyo Institute of Technology, Real World Computing Partnership, Kyoto University and NTT Software Inc. Ninf is built using C and C++ languages. Administrators and application vendors have to implement a secure communication system because the communication between a client and the server is achieved by standard TCP/IP socket connection [13]. In an heterogeneous environment, Ninf uses the Sun XDR data format as a default protocol. Special features implemented in clients, such as interactive data visualization, I/O of data, etc. helps user to use system easily. Ninf makes scheduling decisions based on the dependencies of the input data: any groups of remote calls are analyzed for dependency relationships, and those that may be executed without waiting for another call to finish are immediately sent off to a remote resource. Users cannot take part in scheduling decisions. Although a programmer can define blocks of related function calls, the actual selection of a remote resource and the migration of data is handled by the Ninf Metaserver. When a server fails to complete a task, the task is simply rescheduled to a new resource. Ninf in-

Table 1: Easy accesibility of NES environments by the stakeholders: (\checkmark) - easy to adopt, (*) - fairly adoptable, (+) - amelioration is required

NES environments	Adminsitrator	Application Vendor	Developer	Owner	User
DIET	+	\checkmark	+	+	*
NeOS	+	+	+	+	+
NetSolve	+	\checkmark	+	+	*
Nimrod	+	+	\checkmark	+	+
Ninf	+	*	+	+	+
PUNCH	+	\checkmark	+	+	*
WebCom	*	+	\checkmark	+	+

tegrates with the Condor system for checkpointing to enable fault tolerance for computation.

4.6 PUNCH

Purdue University Network Computing Hubs [11] is developed at Purdue University, USA. PUNCH is a demand-based network-computing system. Tools do not have to be written in any particular language, and access to source and/or object code is not required. The simulation is started via a browser interface. PUNCH components communicate through TCP/IP and HTTP protocols. All the interfaces to the PUNCH system is via WWW enabled browsers. These components use access codes and hash functions to authenticate component, passwords and users identity.

PUNCH system's front end have logical accounts one per user. PUNCH can handle single point failures with the help of management unit via scalability. Fault tolerance is achieved through distributing the software resources among appropriate number of management units. If management unit crashes, requests can not be executed by PUNCH.

4.7 WebCom

WebCom [12] has developed at Centre for Unified Computing, Cork, Ireland. WebCom-G is implemented in Java and is compatible with Win 9x/XP and all flavors of Unix and Linux. The main goal of the WebCom-G project is to "hide the Grid". WebCom-G is an execution platform for applications expressed in Condensed Graphs. WebCom is developer friendly due to its modular and pluggable components. Developers can easily add and extend the features and functionality of WebCom by using these components. However, users and application vendors should have a good background knowledge of Condensed Graphs. WebCom employes the keynote trust management system for its security

and provide layered checkpointing. WebCom components are changeable and promotionable depending on the availability of machines and the number of submitted application. This feature is problematic for administrator in terms of security.

Table 1 summarizes the adaptability of above mentioned NES environment with respect to the perspective of administrators, application vendor, developers, owners and users.

5 Moving beyond NES environments

In addition to these NES environments, other middleware infrastructures are available to execute remote jobs over the Grid.

Some of these middleware are based on cycle stealing concept unlike NES environments, which are dedicated servers. For example Condor [14], is a job management system for compute-intensive jobs. XtremWeb [5] is intended to distribute applications over a set of hosts using a cycle stealing scheme and particularly focuses on multi-parameters applications which have to be computed several times with different inputs and/or parameters, each computation being fully independent from each other.

Some systems that are dedicated but do not provide a scheduling system, job management services, queuing mechanisms as provided by NES environments including the following. The Globus [10] system is a dedicated Grid system which falls into this category. Globus is a Grid toolkit to build computational Grids and Grid-based applications.

Some systems have dedicated servers with code that will be needed to analyze input data. So only data has to be transferred from the user to the server.

6 Conclusion

This paper has focused on the important needs of NES environment stakeholders from their point of view. An outline of an ideal NES environment on the basis of their requirements has been presented, so as to provide a basis for the development of a more efficient NES environment or to ameliorate the existing NES environments.

If NES environments are going to meet the goals outline in Section 3, then important issues such as security and dynamic visualization have to be better supported. Security is important for many reasons: resources typically belong to multiple owners, users worry about having their data exposed to a public network, and there is a real need to prevent malicious use of valuable shared resources. Visualization keeps users in contact with their executing code and promotes a feeling of transparency. It can also be important in the implementation of computation steering techniques and in the development of fault tolerant algorithms.

References

- [1] D. Abramson, R. Sasic, J. Giddy, and B. Hall. Nimrod: A tool for performing parametrised simulations using distributed workstations. The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
- [2] A. Amar, R. Bolze, A. Bouteiller, P. K. Chouhan, A. Chis, Y. Caniou, E. Caron, H. Dail, B. Depardon, F. Desprez, J.-S. Gay, G. Le Mahec, and A. Su. Diet: New developments and recent results. In *CoreGRID Workshop on Grid Middleware (in conjunction with EuroPar2006)*, Dresden, Germany, August 28-29 2006.
- [3] D. C. Arnold, H. Casanova, and J. Dongarra. Innovations of the NetSolve Grid Computing System. *Concurrency and Computation: Practice and Experience*, 14(13-15):1457-1479, 2002.
- [4] A. S. Artelys, E. D. Dolan, J. P. Goux, R. Fourer, and T. S. Munson. Kestrel: An interface from modeling systems to the NEOS server. Technical report, 2003.
- [5] F. Cappello, S. Djilali, G. Fedak, T. Héroult, F. Magniette, V. Néri, and O. Lodygensky. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Comp. Syst.*, 21(3):417-437, 2005.
- [6] E. Caron and F. Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 2006.
- [7] H. Casanova and J. Dongarra. NetSolve: a network server for solving computational science problems. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 40, 1996.
- [8] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225-267, 1996.
- [9] P. K. Chouhan. *Automatic Deployment for Application Service Provider Environments*. PhD thesis, cole normale supérieure de Lyon, France, Sept. 2006.
- [10] I. Foster and C. Kesselman. Globus: A meta-computing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115-128, Summer 1997.
- [11] N. H. Kapadia and J. A. B. Fortes. PUNCH: An architecture for web-enabled wide-area network-computing. *Cluster Computing*, 2(2):153-164, 1999.
- [12] J. P. Morrison, J. J. Kennedy, and D. A. Power. WebCom: A Web-Based Distributed Computation Platform. Proceedings of Distributed computing on the Web, Rostock, Germany, June 21 - 23, 1999.
- [13] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Nin: Network based Information Library for Global World-Wide Computing Infrastructure. In B. Hertzberger and P. Sloot, editors, *High-Performance Computing and Networking (HPCN'97 Europe)*, volume 1225 of *Lecture Notes in Computer Science (LNCS)*. Springer Verlag, Vienna, Apr. 1997.
- [14] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.