Grid-Enabled Earth System Models

S. Zhou, B. Womack, G. Higgins
Northrop Grumman IT /TASC, 4801 Stonecroft Blvd., Chantilly, VA 20151

Abstract- **Grid Computing is quickly being adopted in both commercial and academic environments. It has been used in some areas within the scientific community, but, has not been widely accepted within the Earth science community. In this paper we will discuss a combination of new technologies including Lambda networks, Grid Computing, the Earth System Modeling Framework (ESMF), and the Common Component Architecture (CCA) that together are providing promising opportunities to effectively apply Grid computing in the Earth science domain. We will describe a prototype distributed Earth system model that is being developed using standard Grid service technology and discuss how advances in technology are enabling practical application of Grid computing in this environment.**

I. INTRODUCTION

Typical Earth system models consist of several complex model components coupled together through data exchanges. They require large amounts of data for initialization and may produce an even greater amount during execution. Ideally, an Earth system model would like to have the best combination of components available in the community and to execute on the platform that those components were optimized for. For validation, an Earth system model may also need to compare results using components from different institutions. However, efficiently distributing the data required by model components and managing their execution on multiple computer systems have been two major barriers to the development of distributed Earth system models.

In the last few years, several relevant technology advances have been made: Lambda networks, Grid Computing, Earth System Modeling Framework, and Common Component Architecture. Lambda networks provide transparent end-to-end dedicated bandwidth for fast delivery of huge amounts of data [1, 2]. A Lambda network is being constructed to connect institutions across the U.S. Grid computing integrates networking, communication, computation and information to provide a computation and data management capability that forms a virtual platform for information [3]. Recently the Open Grid Services Architecture (OGSA) has been proposed to standardize Grid Computing. Grid-enabled applications are being demonstrated. The Earth System Modeling Framework (ESMF) is a community framework for Earth systems, funded by NASA's ESTO/CT project [4]. Its goal is to facilitate model coupling and to make models interoperable across organizations. The ESMF software consists of a superstructure (coupling) and an infrastructure (data structures and utilities). DOE also supports the Common Component Architecture (CCA) project defining a minimal set of standard interfaces for a high-performance component framework [5].

ESMF provides a component framework designed for the Earth system community. The current combination of increasing availability of high-speed dedicated bandwidth, standards for Grid applications, and high-performance component frameworks for Earth system models provide an important opportunity to evaluate how the ESMF architecture and its compliant applications can be extended to work in a Grid Computing environment. The current ESMF implementation is directed at running coupled model components on a local computer. Consequently, to couple ESMF compliant model components from different institutions, the source codes and their supporting environment such as input data and libraries have to be physically ported to one computer. Porting the source code for a complex Earth system model to a new machine is only the beginning. To be used operationally, the model must then go through a rigorous validation process to ensure that the porting process did not introduce any errors into the model. It may also need to undergo a performance optimization process to allow it to run efficiently on the new computer.

ESMF's component design encapsulates an Earth system model component's implementation behind a standard interface and provides a handle to access the component. This design facilitates interaction among model components and conforms in principle to the design of a Grid service. To explore this issue, we have Grid-enabled an ESMF-CCA Prototype 1.0 [6, 7] based on CCA's Ccaffeine framework and the ESMF design. XCAT [8], a CCA compliant framework, is chosen to replace Ccaffeine since XCAT allows for CCA components to be compatible with Open Grid Standard Interface (OGSI) specification, which enables CCA components to be accessible via standard Grid clients.

In this paper, we will describe the ESMF-CCA Prototype 2.0 and how its technology can be used to support Earth system models on the Grid.

II. DESCRIPTION OF ESMF-CCA PROTOTYPE 1.0

Coupling model components in an extensible and flexible way is a very challenging and domain-specific problem. CCA components interact by adhering to the Uses-Provides design pattern. This means that each component publishes the functionality that it allows other components to access. These published methods are known as *Provides Ports*. Each component also publishes the functionality that it needs to have other components perform for it. These published methods are known as *Uses Ports*. Conceptually a 'port' can be thought of as a contract between components of a system. It is the equivalent of Java interfaces and pure abstract class definitions in C++. The CCA framework includes one additional type of port. The 'go' port is the starting point for executing systems of components. Driver components implement a '*go'* port which is used for scheduling and controlling the running sequence of components. A CCA compliant framework, like Ccaffeine and XCAT, is responsible for connecting and managing ports. For example, a component, UseAtm, with a *Uses Port* of name Atm1 and type ESM, can be connected to another component, ProvideAtm, with a *Provides Port* of name Atm2 and the same type, ESM (see Figure 1).

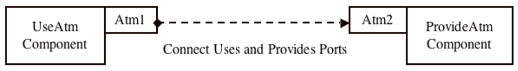


Figure 1 Illustration of coupling two components using CCA

A typical Earth system model component such as an atmospheric model has three general functions that it must perform: initialization, run, and finalization. An initialization routine provides the functionality to prepare a model for simulation and includes initialization of parameters and boundary conditions. A run routine provides the functionality to allow the model simulation to manage its time-evolution. A finalization routine provides the functionality to safely shut down operations, clean up memory, and close any open files. ESMF's component model design requires an ESMF-compliant component to provide implementations for these standard functions and the ESMF superstructure is responsible for mapping these routines to the standard interfaces that an ESMF application component (driver) expects to call. In addition, ESMF provides a standardized, self describing, format for data exchange among model components through the ESMF_State data type.

A key difference between the CCA component environment and the ESMF component environment is ESMF's requirement for the user to supply additional standard methods (e.g. initialization, run and finalization) beyond those required for registering the component in the framework. ESMF provides a standard component adapter class, ESMF_Comp Component that maps the standard component interface to the functionality implemented by the model developer's code. CCA is not application-specific and does not require any additional methods or standard interfaces.

Our ESMF-CCA Prototype 1.0 utilizes CCA's Uses-Provides design pattern to create and couple those CCA components that use the ESMF standard component interfaces described above. This allows ESMF-compliant components to run within a CCA component framework. CCA's Uses-Provides design pattern has been successfully used for coupling various components in a non-intrusive and flexible way.

III. DESCRIPTION OF ESMF-CCA PROTOTYPE 2.0

Ideally, an Earth system model would like to have the best combination of components available in the community and to execute on the platform that those components were optimized for. However, efficiently distributing the data required by model components and managing their execution on multiple computer systems in a standard way have been two major barriers to the development of distributed Earth system models.

XCAT [8], a CCA compliant framework, has been developed to enable CCA components to be compatible with the Open Grid Standard Interface (OGSI) specification. The OGSI enables CCA components to be accessible via standard Grid clients. Basically, XCAT employs the Remote Method Invocation (RMI) mechanism implemented by XSOAP [9] to allow communication and control among local and remote components.

Merging the two standards (CCA and OGSI), XCAT uses an approach where a component is modeled as a set of Grid services. In the following section, we will use the XCAT3 [10] framework, implemented in Java, to illustrate some of key features beyond CCA's Ccaffeine framework:

- XCATComponentID: an interface that extends the CCA's ComponentID. There are a few methods used internally by the XCAT3 framework related to composing components from a remote *Provides Port*.

- Builder Service: a CCA port implemented by CCA compliant frameworks that allows a component to dynamically create and compose other components into an application. Some methods exposed by the Builder service for component lifecycle purposes are createInstance for creating an instance of a component, and destroyInstance for eliminating the component instance from the scope of the framework. Since XCAT3 is a distributed component framework, its createInstance is capable of creating component instances on remote locations using the Globus GRAM [11] protocol provided by the Java CoG Kit [12].

- Scripting Interface: XCAT3 provides an interface to the Builder service using Jython scripts. The Jython API available to the user closely mirrors the API provided by Builder services. The Ccaffeine framework used in ESMF-CCA Prototype 1.0 employs a custom-design script to compose local components. In addition, a GUI is also provided for local component composition within the Ccaffeine framework. The GUI tool is very helpful for component development and multi-level component composition.

ESMF-CCA Prototype 2.0 essentially replaces CCA's Ccaffeine framework with XCAT3 while keeping ESMF's three standard methods, Initialize, Run, and Finalize, and the standard data type, ESMF_State, for exchanging data.

IV. A CODE TO TEST ESMF-CCA PROTOTYPE 2.0

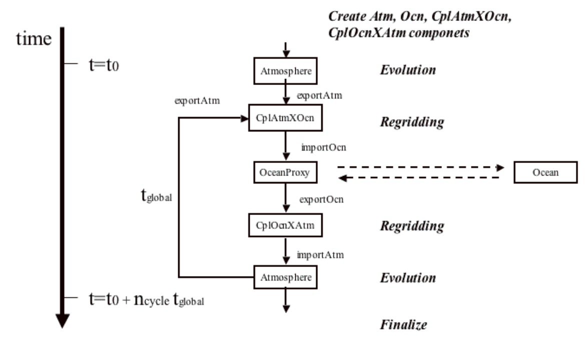
To explore the compatibility issues between ESMF and CCA and test the ESMF-CCA Prototype 1.0, we designed and developed a 2D Simple Coupled Atmosphere-Ocean Model (SCAOM) [7]. SCAOM is computationally similar to, but physically simpler than, the Coupled Shallow Water Model [13]. The atmosphere model component uses a coarse rectangular grid while the ocean model component uses a fine grid. Each model component consists of a forward finite-difference advection scheme and periodic boundary conditions along the x direction. Data is exchanged at the overlapped boundary along the y direction. In addition, each model component can use a different independent timestep. To couple an atmosphere with an ocean model component, we also developed two simple couplers: one for exchanging data from the atmosphere to the ocean and another one for exchanging data from the ocean to the atmosphere. Basically, the couplers transform data from one grid to another grid. The code is written in C++ since CCA's Ccaffeine framework supports C++ naturally.

XCAT3 is written in Java. It currently provides a complete Java API with a C++ API to be released soon. We are in the process of converting the SCAOM code from C++ to Java. At the time of writing this paper, we have developed a stub code, SSCAOM, essentially supporting the interfaces of our original SCAOM code.

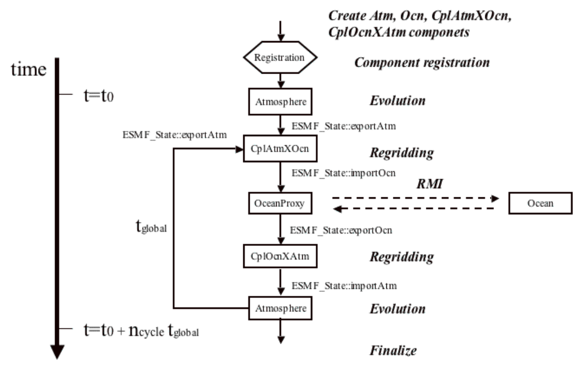
V. RESULTS

A typical sequential coupling sequence between an atmosphere model and a remote ocean model is depicted in Figure 2a, where t_{global} is the time advance in one coupling cycle and n_{cycle} is the number of coupled cycles. Without a framework like XCAT, all the components in Figure 2a including the ocean proxy component would need to be implemented by the model developer. This means that the developer would need to be familiar with network programming with socket, Remote Procedure Call (RPC), RMI, or web services, etc. But the XCAT framework used by ESMF-CCA 2.0 automatically provides a local proxy component for the remote ocean that is hidden from the user. This proxy component handles all of the RMI necessary to execute the remote component and is one of its most user-friendly features.

At the beginning of a simulation, the framework creates each component. In the ESMF-CCA Prototype 2.0, we still use CCA's setService utility to register the components of atmosphere, ocean, coupler from atmosphere to ocean (CplAtmXOcn), and coupler from ocean to atmosphere (CplOcnXAtm). The model components exchange data through coupler components by passing import and export states that are of type ESMF_State (see Figure 2b). The atmosphere component first *provides* its data at its boundary, exportAtm, to the coupler, CplAtmXOcn. CpleAtmXOcn *uses* exportAtm, transforms it into importOcn with an interpolation routine, and *provides* importOcn to the ocean component. With importOcn, the ocean components run its finite-difference advection scheme for $n{*}t_{Ocn}$ timesteps while satisfying its periodic boundary condition in the x direction. After that, the ocean component *provides* its data at the boundary, exportOcn, to the coupler, CplOcnXAtm. The coupler, CpleOcnXAtm *uses* exportOcn, transforms it into importAtm, and *provides* importAtm to the atmosphere component. Similar to the ocean component, the atmosphere component *uses* importAtm, runs its finite-difference advection scheme for $m*t_{Atm}$ timesteps while satisfying its periodic boundary condition in the x direction. Then, the atmosphere component *provides* its data at the boundary, exportAtm, to the coupler, CpleAtmXOcn. This completes a coupling cycle with the time advanced, t_{global} ($=n*t_{Ocn} + m*t_{Atm}$).



(a)



(b)

Figure 2 (a) A sequential coupling between an atmosphere and a remote ocean model component. (b) The flow diagram (a) as implemented in the ESMF-CCA Prototype 2.0.

To test the ESMF-CCA Prototype 2.0, a Java stub code, SSCAOM, supporting the interfaces of a 2D simple coupled climate model, is integrated into the ESMF-CCA Prototype. SSCAOM allows a user to have the following options: (1) different grid resolution for Atmosphere and Ocean model components to test regrid functionality in the coupler components. (2) different number of iterations for Atmosphere and Ocean model components to mimic their corresponding production codes.

In the ESMF-CCA Prototype 2.0, each component is developed, compiled as a java class (.class), and stored in an individual directory. With a Jython script, a user lists *Provides* components and *Uses* components, specifies their locations (e.g., file directory and computer machine), and then chooses one of three mechanisms to handle creation of the components: local, ssh, or gram (which uses the Globus GRAM). After that, live instances of the components are created on the target computer machine and the *Provide Port* and *Uses Port* of the components are connected and the execution of the program is started by invoking the go method of the XCATGoPort implementation of the *Uses* component.

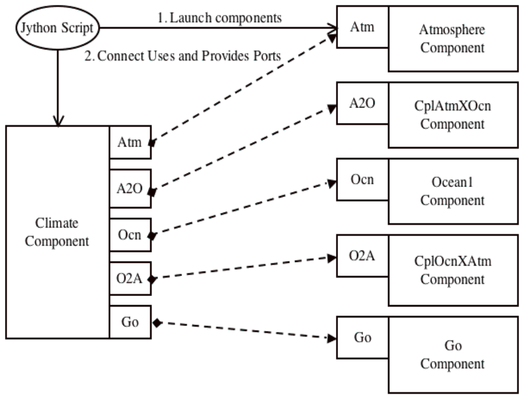


Figure 3 Composition of components without a third-party registry service. A Jython script is used to launch local and/or remote components. After that, the Builder Service is used to couple components by connecting their *Uses* to *Provides* Ports.

We have placed an Ocean component on a remote machine and used the ssh method to create and execute the remote component. With the proxy RMI technology used in XCAT, the procedure is similar to running a simulation with the Ocean component on the local machine. The difference is apparent when specifying the location of the remote component and determining the mechanism for launching the component in a Jython running script. The process is illustrated in Figure 3.

When composing an application without using a registry service as shown in Figure 3, a user has to know in advance, all the components to be likely used. The locations and methods of invocation for each component are written into the code. That requirement can be simplified by using a Registry Service. A Registry that is OGSI-compliant, persistent, and implemented by a third party, can be used to register locators of those components with *Provides Port*. Then a user can use a Jython script to query the Registry for the availabilities, capabilities and the location of the component. The script can select the components that it wishes to use and connect them via Builder Services. For example, a user can pick one of the available Ocean components with the necessary component interface before starting a simulation (see Figure 4). This capability to select and use similar model components from different providers is a key feature that is needed in the Earth science community. It enables the user to quickly configure and run experiments that examine the behavior of coupled model systems. Climate modeling, in particular, can use this feature to explore and understand the factors that contribute to present climate predictions since different models can produce significantly different results due to even small variations in initial conditions and parameters. Ensemble studies that include multiple models components of the same type could also be easily constructed.

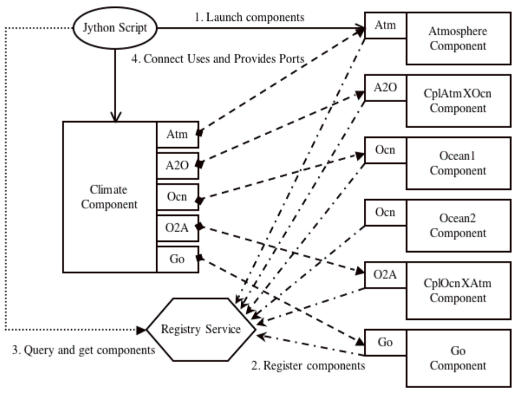


Figure 4 Composing components with a third-party Registry Service.

VI. DISCUSSION

By testing the ESMF-CCA Prototype 2.0 with a stub of a simple coupled climate model mentioned above, we found that a Grid-enabled Earth system can be supported by combining ESMF with CCA. ESMF and CCA complement each other nicely: ESMF standardizes the interfaces to Earth system model components and provides utilities and standard data formats for building and coupling these components together. This combination considerably facilitates coupling models via the Grid. CCA provides a flexible method for controlling and managing components and their coupling. CCA's component model makes a CCA component easily transformed into a Grid service.

ESMF's component model is based upon an interface wrapper that provides a common set of standard methods. These interface methods are connected to user provided implementation code through a C function pointer table contained within the ESMF_COMP class. The user implementation for ESMF component methods are usually provided as a Fortran90 module containing functions that correspond to the standard component methods. However, the function pointer table entries can be assigned to any function that matches the signature of the expected methods.

We are currently investigating the use of a Grid-enabled ESMF proxy component as shown in Figure 5. It uses the external interface that the ESMF component wrapper expects for the standard user provided functions, but, internally the proxy component redirects the method calls to a Grid-service that runs on another machine.

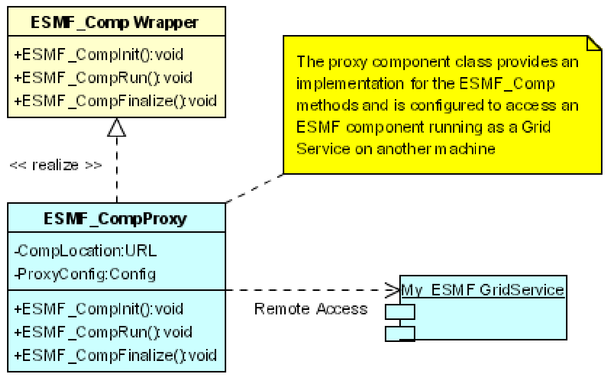

ESMF_Comp Wrapper
+ESMF_CompInit():void
+ESMF_CompRun():void
+ESMF_CompFinalize():void
The proxy component class provides an implementation for the ESMF_Comp methods and is configured to access an ESMF component running as a Grid Service on another machine
<< realize >>
ESMF_CompProxy
-CompLocation:URL
-ProxyConfig:Config
+ESMF_CompInit():void
+ESMF_CompRun():void
+ESMF_CompFinalize():void
Remote Access
My ESMF GridService

Figure 5 Proxy component class diagram

The Proxy component is configurable to allow the user to specify information related to the component's actual location, where its initialization files reside, where its output should go, etc. These options allow the proxy to be reused for connecting to alternative Grid-service providers as shown in Figure 6.

The XCAT3 framework provides a relatively simple method to publish an ESMF component as a standard Grid-service. The addition of an ESMF compliant Proxy component will make it possible for ESMF application developers to quickly connect an ESMF component that has been published as a Grid service through XCAT3.

ESMF application developers will be able to avoid the painful process of porting and revalidating the model components that they want to include in their Earth system modeling application. Currently, when a new coupled Earth system model is created, it takes years of development effort before any results can be evaluated. Using the Grid services and proxy model described here, the time required to make the initial connections necessary will be greatly reduced.

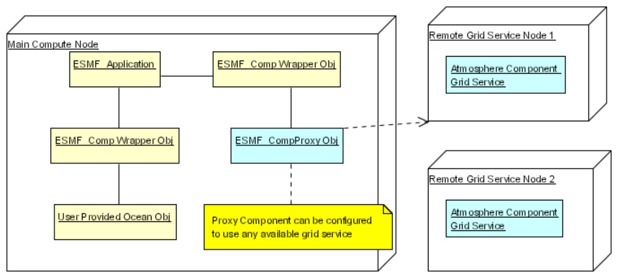

Main Compute Node
ESMF_Application
ESMF_Comp Wrapper Obj
ESMF_Comp Wrapper Obj
ESMF_CompProxy Obj
User Provided Ocean Obj
Proxy Component can be configured to use any available grid service
Remote Grid Service Node 1
Atmosphere Component Grid Service
Remote Grid Service Node 2
Atmosphere Component Grid Service

Figure 6 ESMF proxy component accessing a remote component using grid services

One frequent objection to using the Grid for Earth system modeling applications is that they tend to produce and consume too much data. An average user can only expect to see 10-50 Mbps throughput using the Internet2 even though it is a 10 Gbps network [14]. Under these conditions, the time necessary to move large input and output data sets that are commonly produced make it unrealistic to consider the use of distributed computing in the Earth system modeling community. However, the primary limitation for throughput using the internet and internet2 is not lack of bandwidth. It is the design of the TCP protocol that provides the foundation for most data transfer protocols commonly used. For example in October 2003, UIC's National Center for Data Mining (NCDM) and Laboratory for Advanced Computing flashed a set of astronomical data across the Atlantic at 6.8 gigabits per second—6,800 times faster than the 1 megabit per second effective speed that connects most companies to the Internet. In the Oct. 10 test run, 1.4 terabytes of astronomical data were transmitted from Chicago to Amsterdam in 30 minutes using UDT (UDP-based Data Transport), a new protocol developed by the NCDM. Moving the same amount of data using today's standard protocol for data transfers, TCP, would take 25 days [15]. Hence, data movement is no longer the large obstacle that it used to be when considering distributed Earth system models. New transport protocols such as UDT are removing these barriers and even greater bandwidth will soon be available. The National Lambda Rail is currently under development and will provide multiple dedicated Gbps links between the major research centers across the United States [14].

The High End Computing Network (HECN) Group at Goddard is currently implementing a local Lambda test network in preparation for connecting to the National Lambda Rail in the Fall '04. This test network will provide multiple 10 Gigabit dedicated lambdas between two Linux clusters located at opposite sides of the GSFC facility. We expect to extend our prototype system to couple two of NASA's ESMF compliant models [4] (e.g., one land model and one atmosphere model) over the Lambda network using two parallel computers at NASA's Goddard Space Flight Center (GSFC).

VII. CONCLUSION

We have developed an ESMF-CCA Prototype 2.0 by updating our ESMF-CCA Prototype 1.0 with the features of Grid computing. Preliminary testing on the prototype shows that Earth system models can be accessed remotely with Grid technology in a user-friendly way.

ACKNOWLEDGEMENT

This is project is supported by the NASA Earth Science Technology Office Computational Technologies Project. We would like to thank Sriram Krishnanand and Madhusudhan Govindaraj for technical support in using the XCAT framework and Dennis Gannon and Randall Bramley for helpful discussion.

REFERENCES

[1] Prototyping Tomorrow's Optical Cyberinfrastructure, http://www.calit2.net/briefingPapers/optiputer.html
[2] GTP: Group Transport Protocol for Lambda-Grids, http://www.optiputer.net/publications/articles/CHIEN-GTP_CCGrid2004.pdf
[3] F. Berman, G. Fox, and A.J.G. Hey, "Grid Computing", Wiley, 2002
[4] Earth System Modeling Framework, http://www.esmf.ucar.edu/
[5] Common Component Architecture, http://www.cca-forum.org/
[6] ESMF-CCA Prototype, http://ct.gsfc.nasa.gov/esmf_tasc/index.html
[7] S. Zhou et al., "Prototyping of the ESMF Using DOE's CCA", NASA Earth Science Technology Conference 2003
[8] XCAT, http://www.extreme.indiana.edu/xcat/
[9] XSOAP, http://www.extreme.indiana.edu/xgws/xsoap/
[10] Sriram Krishnan and Dennis Gannon, "XCAT3: A Framework for CCA Components as OGSA Services", Proceedings of HIPS 2004, 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments. April 2004.
[11] K.Czajkowski, I. Foster,N. Karonis,C. Kesselman, S.Mar tin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. InIPPS/SPDP98,

Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[12] G. von Laszewski, J. Gawor, S. Krishnan, and K. Jackson. Grid Computing: Making the Global Infrastructure a Reality, chapter 25, Commodity Grid Kits - Middleware for Building Grid Computing Environments. Wiley, 2003.

[13] C. Cruz, M. Kelly, M. Clausen, S. Zhou, and G. Higgins, "Coupled Ocean-Atmosphere Shallow Water Model,'' TASC Internal Report (2001).

[14] J. Patrick Gary, "NASA Goddards Vision for 10 Gigabit Ethernet" (March 2004)

[15] Speed Record Set at UIC, http://access.ncsa.uiuc.edu/Releases/03Releases/10.14.03_Speed_Reco.html