Developing service-oriented applications in a grid environment

Experiences using the OPeNDAP back-end-server

Jose Garcia · Peter Fox ·
Patrick West · Stephan Zednik

Received: 10 September 2008 / Accepted: 18 December 2008 / Published online: 20 January 2009
© Springer-Verlag 2009

Abstract The Open-source Project for a Network Data Access Protocol (OPeNDAP) software framework has evolved over the last 10 years to become a robust, high performance, service oriented architecture for the access and transport of scientific data from a broad variety of disciplines, over the Internet. Starting with version 4.0 of the server release, Hyrax, has at its core, the Back-End Server (BES). The BES offers ease of programming, extensibility and reliability allowing added functionality especially related to high-performance data Grids, that goes beyond its original goal of distributed data access. We present the fundamentals of the BES server as a component based architecture, as well as our experiences using the BES

to distribute data processing in Grid-oriented intensive parallel computational tasks.

Keywords Computational power · OPeNDAP back-end-server · Service-oriented applications · Grid computing

Introduction

A common theme in solving large, computationally intensive problems is to break the system into multiple components and distribute them among a collection of computers, each contributing towards the aggregate computational power. The type of configuration, protocols, resources and interfaces used to define a distributed computational unit leads to entities such as clusters, Grids and multiprocessor machines. In this paper we focus on developing applications in Grid platforms. The term Grid has many definitions, the most common being the view presented by Foster (2002) in his paper “What is the Grid? A Three Point Checklist” (Foster 2002). In that definition, three elements are required:

1. Non-centralized control.
2. Use of standard, open, general-purpose protocols and interfaces.
3. To deliver nontrivial qualities of service.

For our purpose, only the second and third point are relevant, as the experiences presented in this paper are useful for environments where the Grid elements are under either centralized or decentralized management. In other words, our interest is to present the results of developing a high performance architecture

Communicated by: H.A. Babaie

J. Garcia (✉)
Computational & Information System Laboratory (CISL),
National Center for Atmospheric Research (NCAR),
P.O. Box 3000, Boulder, CO 80307, USA
e-mail: jgarcia@ucar.edu

P. Fox
Rensselaer Polytechnic Institute, 110 8th Street,
Troy, NY 12180-3590, USA
e-mail: pfox@cs.rpi.edu

P. West · S. Zednik
High Altitude Observatory (HAO), National Center
for Atmospheric Research (NCAR), P.O. Box 3000,
Boulder, CO 80307, USA

P. West
e-mail: pwest@ucar.edu

S. Zednik
e-mail: zednik@hao.ucar.edu

which is service (data and computation) oriented, that supports component-based software engineering, and which relies of a common set of data access standards.

In deploying a Grid service, the ensemble of elements works together under an encapsulation that conforms the atomic abstraction (Ghezzi et al. 1991) of what the service is, what it can do, and how is accessed. To the user, the service is an entity that behaves like a unit and that is accessed via a standard set of calls. To the developer, a service is a collection of things that must interact in unison hiding all the details of the joins. In the latter, a good middleware for forming the joins is paramount to the higher goal of encapsulation and abstraction, and much work has been dedicated to achieve this goal (Emmerich 2000). Middleware for deploying services roughly falls into two categories and partially overlapping viewpoints (Mathiassen 2000): the component based architecture and the process based architecture. In the former, the individual elements are components which are hosted by an application server or application environment whereas in the latter, the elements are processes (executables) joined together by input-output redirection, such as those that depend on XML (http://www.w3schools.com/xml/xml_what.asp, http://www.w3schools.com/soap/soap_intro.asp) based data, fork calls, exec calls, pipes, etc. (Gray 1998). The component based architecture is generally perceived as more complex to program because its requirements for interfaces and how those interfaces are implemented by each component. Furthermore, the run-time environment in which components are hosted demand from software engineers the consideration of many non-trivial programming task such as memory management. In exchange for these added demands on programmers, component based software offers higher performance as communication among the different pieces within and application is memory based and state is easily maintained. An example of component based architecture is the deployment of WWW sites using Tomcat and frameworks such as Spring (<http://www.springframework.org/>). The process based architecture is easier to deploy because it basically requires for programmers to match the input/output among the different pieces, so that the flow of information is consistent leaving aside considerations on how the processes involved execute their respective tasks. Evidently this architecture has disadvantages in terms of what can be designed by merely managing the flow of information but its simplicity makes it attractive for systems in which high performance is not critical. For example a web site that uses fork calls to execute a relational database (e.g. MySQL; www.mysql.com)

client to be able to access a relational database provides a simpler way to create pages which are data driven yet will not scale well for many concurrent users.

The OPeNDAP Back-End-Server (BES) (Potter et al. 2006) is an application server that exemplifies a component based architecture. In the present context, we define the term component by its characteristics (Szyperski 1998):

- A component is a unit of independent deployment.
- A component is a unit of third party composition.
- A component does not have persistent state.

In the BES, components are written according to the BES data handler specification (see “[The OPeNDAP back-end-server \(BES\), design and implementation](#)”), which implies that third parties can develop the components (referred to as data handlers) independently and can submit their components for execution under the BES environment.

The “[The BES, history and background](#)” presents background work and a historical perspective of the development of the BES. The “[The OPeNDAP back-end-server \(BES\), design and implementation](#)” section presents details of the BES as an application server and its characteristics as a component based application. The “[Example of scientific application deployed with the BES](#)” section presents one scientific application component for the BES that goes beyond the original scope of data access. The “[Conclusions](#)” section provides a summary and future directions.

The BES, history and background

Around the year 1993, the Distributed Oceanographic Data System (DODS, <http://opendap.org/archive/design/data-delivery-arch/>) introduced the Data Access Protocol (DAP, <http://www.opendap.org/pdf/ESE-RFC-004v1.1.pdf>) as a way for clients to access distributed datasets. Quickly, use of the DAP protocol progressed beyond oceanography into other scientific communities, and after completion of the DODS project, the organization that provided core software (client and server) implementing the DAP was later named the Open-source Project for a Network Data Access Protocol (OPeNDAP; www.opendap.org) with the intent of it becoming an open standard. “Although OPeNDAP was originally designed and developed by oceanographers and computer

scientists for oceanographic data, there is nothing in the design of OPeNDAP that constrains its use to oceanography. Indeed, it has been adopted by the High Altitude Observatory community and is being considered by segments of the meteorological and space science communities (<http://opendap.org/faq/whatIsDods.html>).

Under its first web-based server implementation, OPeNDAP used the state of the art technology available at the time, the Common Gateway Interface (CGI, <http://www.w3.org/CGI>) environment. Under this design, the different data formats supported for OPeNDAP access were independent executables invocable through the common DAP URL standard (DAP-Format, <http://www.opendap.org/user/guide-html/url-parts.html>).

For certain applications such as those we consider here, the CGI based implementation had obvious shortcomings; it tied OPeNDAP to the Hyper Text Transport Protocol (HTTP), had obvious limitations in terms of performance and it restricted the possibilities for high performance applications as it lacked support of advanced memory management, distributed computation, etc.

In the year 2000, the Earth System Grid project (ESG, <http://www.earthsystemgrid.org/>) adopted OPeNDAP to support its mission because it offered excellent support for netCDF, which is the prevalent data format for output in climate model simulations. Although OPeNDAP as a protocol satisfied the needs of the ESG communities, its implementation as a server lacked the high performance support required by ESG for distributing its massive archive of data. At the time, an individual request-response could refer to a multi-gigabytes file (Allcock et al. 2001). For this reason, an ESG specific version of the OPeNDAP server was developed by the ESG team. This branch of the main OPeNDAP software became known as OPeNDAP-g where the g was added to indicate the gridded nature of the OPeNDAP branch for ESG.

The OPeNDAP-g server implementation was designed as a multi-tier architecture allowing data access protocol component to be a stand-alone process because it offered three main characteristics;

- Protocol independence. The server could be accessed from HTTP connectors, gridFTP connectors, etc.
- Own address space. It afforded the programmers greater flexibility for high performance development.
- Stateful connections.

In the year 2006, the OPeNDAP core developers commenced an effort to merge the development branch of OPeNDAP-g with the evolving needs of the larger OPeNDAP community. Specifically, the OPeNDAP-g application server replaced the OPeNDAP CGI based server. In turn, support for other formats besides netCDF were added. During this effort, the internals of the OPeNDAP-g server were rewritten under a component architectural approach. This implementation became the BES. The BES, together with the OPeNDAP Lightweight Front-end Server (OLFS, <http://opendap.org/download/olfs.html>) now comprise the OPeNDAP 4 server architecture also known as Hyrax (Potter et al. 2006).

The OLFS is written in Java under the servlet specifications and runs under Apache Tomcat. The OLFS communicates with the BES to provide data and catalog services to clients. The OLFS implements the DAP2 protocol and supports some of the new DAP4 features. The OLFS is responsible as well for implementing the Thematic Realtime Environmental Distributed Data Services (THREDDS, <http://www.unidata.ucar.edu/projects/THREDDS>) catalogs responses, which are satisfied with calls to the BES.

The terms Hyrax and BES are often confused. In this paper, we concentrate exclusively on the BES, leaving aside characteristics that are intrinsic to Hyrax as a whole, such as metadata. The reader is invited to learn further about Hyrax by using the references provided.

The OPeNDAP back-end-server (BES), design and implementation

The BES is a high-performance, component based server framework. This framework can be used to allow data providers to flexibly provide views of their data inventory, means of transforming their data such as through aggregation and/or subsetting, and providing end users with varying data products, such as DAP response types.

To be able to deliver “nontrivial qualities of service”, a typical data request goes through a sequence of stages in which users narrow down the specifics of their request. With this in mind, it was of fundamental importance to consider how the BES design will keep track of *state*. In other words, for the BES to be able to deliver data that is custom processed for each user, as well as for the BES to support server side capabilities, it requires to keep a state reflecting all transactions executed on behalf of a user up to a given point. Since

OPeNDAP was originally based on WWW technologies, consideration was given to mechanisms for session tracking on top of a stateless protocols, similar to those commonly used on modern WWW applications. However, the loose coupling in this client-server model leads to a communication paradigm that is fully realized through messaging in which “state information can be carried and persisted by the messages that pass through the services” (Erl 2005). Parallel to this approach, Java Beans and/or other forms of object persistence could further facilitate maintaining state, but the associated need for either message or object persistence frameworks undermines the simplicity of the connectionless architecture. This lead to the conclusion that the BES will benefit much more from a fully connected client-server model.

To keep session control, the BES uses the Point-to-Point Transfer Protocol (PPT) protocol (PPT, http://scm.opendap.org:8090/trac/wiki/BES_Chunking). PPT is connection-oriented and allows message interchange over a permanently connected channel that keeps clients and servers fully coupled for the duration of the session. In many ways, the BES was modeled after the architecture of popular Relational Database Management Systems (RDMS) as client monitors, client libraries, etc. can stay connected to the server as long as necessary to execute transactions that are dependent on the history of the connection.

The BES is extensible by dynamically loading external modules. Developers can implement modules and have the BES load them in. Modules interact with the BES framework and can interact with other loaded modules. These modules add to, and sometimes modify functionality in, the BES framework. There are already quite a few modules that can be loaded, provided by varying scientific communities such as the Coupling Energetics and Dynamics of Atmospheric Regions (CEDAR) community (CEDAR, <http://cedarweb.hao.ucar.edu>) the Earth System Grid project (ESG, <http://www.earthsystemgrid.org/>), the Community Stokes Analysis Center (CSAC) project (CSAC, <http://www.hao.ucar.edu/projects/csac/>) and other OPeNDAP community members.

In order to allow for external connectivity for client-server systems, the BES runs as a daemon with open TCP/IP and Unix sockets. Clients can connect with the server and interact using the PPT (http://scm.opendap.org:8090/trac/wiki/BES_Chunking) to be able to submit request and to receive associated responses. The BES has a set response pattern to requests. Each step in the pattern is extensible by the dynamically loaded modules. For example, modules can add initial-

ization callbacks that are called during the initialization step. This pattern is:

1. initialization step
2. validating the incoming information
3. building the request plan
4. executing the request plan
5. transmitting the response
6. logging the status of the request
7. metrics reporting on the request
8. termination step

The BES is extensible by providing developers the ability to provide:

1. new data handlers for access to various data formats, for example: Hierarchical Data Format Release 4 (HDF4, <http://hdf.ncsa.uiuc.edu/products/hdf4/index.html>), Network Common Data Form (NetCDF, <http://www.unidata.ucar.edu/software/netcdf>), Flexible Image Transport System (FITS, <http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html>), etc.
2. varying response types (OPeNDAP DAP responses, CEDARS tab separated text format)
3. commands to handle new responses (Commit command to save to disk aggregated data definitions)
4. new request types
5. initialization and termination callbacks
6. exception handling mechanism for errors specific to their module
7. metrics reporting
8. persistent data container and view definition storage
9. varying aggregation engines.

The BES can be deployed to provide Grid services. For example, multiple BES instances acting as computational engines that transform data via software algorithms can be handed different slices of a dataset to operate on, the results being fed back to a controller process where the individual slices can be combined to form the resulting data product essentially working as Single Process, Multiple Data (SPMD) (Quinn 2004). Figure 1 highlights the component architecture of the BES, showing among others, the communication component, the PPT protocol, the DAP component, the data formats netCDF, HDF and FreeForm (http://www.opendap.org/download/ff_server.html) components. All components work together to deliver data as users request it.

Fig. 1 The BES architecture with some third party modules for supporting data access

Example of a scientific application deployed with the BES

In order to illustrate the broad range of applications that can be supported by the extensibility offered by the BES, one example is presented (Garcia et al. 2007).

The Community Spectro-polarimetric Analysis Center (CSAC, <http://www.hao.ucar.edu/projects/csac/>) is a National Center for Atmospheric Research (NCAR) Strategic Initiative that provides a suite of community resource tools for analysis of precision polarization data for remote sensing of magnetic fields in the solar atmosphere. The basic set of numerical libraries developed to support the computations is known as Milne Eddington gRid Linear Inversion Network (MERLIN) (Lites et al. 2007). The aim of this project is to make the basic analysis tools easily accessible to users in the scientific community.

On 22 September, 2006, the HINODE (<http://solarb.msfc.nasa.gov/>) satellite was launched from the Uchinoura Space Center, Kagoshima, Japan. Among the instruments it carries, the Solar Optical Telescope (SOT, <http://solarb.msfc.nasa.gov/mission/ota.html>) performs the acquisition of spectro-magnetic signals used to determine magnetic fields in the Solar Photosphere. In 2006, the spectrograph produced from October to December, 254,198 spectral images with a total volume of 272.7246 gigabytes. In 2007, the spectrograph produced 1,494,749 for a total volume of 2.8919 terabytes. In the year 2008, the spectrograph has produced a total of 202,413 images for a total volume of 252.5415 gigabytes.

In order to be able to compute the inversions, that is, to obtain the magnetic field information from a given spectral signal, each individual spectral image must be

consumed by the numerical engine in MERLIN. The total inversion time of each individual spectral image varies depending on the area of the Sun from which the image was obtained (highly active regions take longer), the number of pixels in the image, the convergence criteria desired by the user, the quality of the initialization, etc. However, our empirical results are that a single spectral image containing 1,000 pixels, running on a AMD Opteron with a 2.4 GHz. clock frequency, takes under average conditions 18 s for inversion, or 0.02 s per pixel. This yields an average computational capacity of 50 pixels per second per processor.

The spectro-polarimeter in the SOT instrument has different modes of operation (SOTModes, http://solar-b.nao.ac.jp/sot_e/sp_e.shtml). The rate at which data is acquired depends on the specific mode of operation, and the speed at which data reaches the operation center on earth depends on the capacity of the transmitter. Furthermore, depending on scientific goals, the SOT instrument could be idle while other of the instruments on board acquires data. All this criteria makes difficult to decide the minimal capacity for inversion in the data grid, however the CSAC management group decided based on heuristics that the inversion Grid must be able to process at least 300 pixels per second. Hence at least six processors like the AMD Opteron 2.4 GHz mentioned in the previous paragraph are required. Furthermore, to allow for experimentation, inversions with non-standard parameters, errors, etc. the base CSAC Grid hardware consist of a total of 20 AMD Opteron 2.4 GHz CPUs. For cases when heavy loads enter the system, the CSAC Grid has the option of using idle cycles from a set of other non-dedicated workstations running Intel Xeon processors.

Given the above requirements, a parallel software solution was designed. To develop the CSAC inversion Grid, we created a BES module containing the MERLIN engine and distributed multiple BES instances among all processors on the Grid. To coordinate the distribution of the inversion of the multiple spectral images, a web based application runs as a front end, with socket connections to the multiple BES. Each of these connections follows the specifications of the PPT protocol with the request-response paradigm as illustrated in the “[The OPeNDAP back-end-server \(BES\), design and implementation](#)” section. The web application acts as a SPMD controller so all necessary task are parallelized using the different BES instances to provide inversions as a Grid service. When inversions starts, the controller looks at the topology definition and engages as many BES as nodes are available on the Grid. In practice, this is done with a table in a

Fig. 2 The high-performance CSAC inversion engine relies on multiple instances of the BES for work distribution

relational database, in which a column simply defines if a node is available or not. This database is controlled by the CSAC Grid administrator so at any given point a minimum of 20 processors or a maximum of 32 are available for work.

Figure 2 shows a schematic of the complete inversion engine as multiple instances of the BES running on different hosts, controlled through a single front-end, implemented as an Apache (Kew 2007) module. All transaction logic with the user is handled by the web server and the heavy data processing is executed on the back end. The Apache server keeps a fully connected set of BES instances that perform tasks on its behalf.

The client provides the guided user interface to the complete system. With a browser, the user can log into the CSAC Grid, select a dataset for inversion and execute the inversion. The details of how the inversion runs in parallel, how the different BES provide services to the CSAC Grid are all encapsulated and abstracted from the user.

To experiment on its capacity, two well known spectral maps containing Solar Active Regions obtained on December 12, 2006 and May 3, 2007 were inverted under different sets of parameters. Under this load, the CSAC Grid scaled almost linearly since the process of *scattering-gathering* the multiple tasks as executed by the controller is minimal with respect of the total cost of the inversions as executed by the MERLIN component in each of the BES instances.

Conclusions

Over the last ~ decade, the OPeNDAP BES server has evolved from its WWW roots into a flexible, component

based system that allows for third-party modules to be integrated into a service based architecture. At its core, now the OPeNDAP BES can deliver data from a variety of formats such as HDF4, netCDF, FITS, etc.

With the example presented we have illustrated how multiple instances of the BES can cooperate to perform tasks in parallel. In the future, as the size of datasets grow, the BES can be customized to work as a Grid service to deliver data with higher performance. Modules that comply with the BES interface standards can be loaded into the BES and can be reached and executed using a set of BES request-response calls.

As the CSAC Grid gains more experience with the production load of regular inversions, we will report on the efficiency and other quantitative benefits of the approach we describe here.

Availability and requirements

The OPeNDAP BES framework can be downloaded from <http://www.opendap.org>. The BES binary software is available at the web site for Solaris, SGI Irix, Linux, Compaq/Digital Unix and Windows. For users interested in building their own BES, the source package provided by the distribution uses the GNU autotools for configuration and build which makes very compatible with most Unix platforms.

For developers, OPeNDAP offers a web site <http://scm.opendap.org:8090/trac> where a wealth of information and the source code is available for the community at large.

References

- Allcock B, et al (2001) High-performance remote access to climate simulation data: a challenge problem for data grid technologies. ACM/IEEE SC 2001 Conference, p 20 (SC)
- Emmerich W (2000) Software engineering and middleware: a roadmap. In Proceedings of the conference on the future of software engineering, pp 117–129
- Erl T (2005) Service-oriented architecture. Prentice Hall, Indianapolis, ch 3
- Foster I (2002) What is the grid? A three point checklist. Argonne National Laboratory & University of Chicago foster@mcs.anl.gov. July 20
- Garcia J, Fox P, Lites P (2007) Parallel analysis of spectropolarimetric signals in heterogeneous grids, using OPeNDAP BES to perform scatter-gather high performance computing. Eos Trans. AGU, 88(52), Fall Meet. Suppl., Abstract IN11B-0466
- Ghezzi C, Jazayeri M, Mandrioli D (1991) Fundamentals of software engineering. Prentice Hall, New Jersey, pp 319
- Gray J (1998) Interprocess communication in unix. Prentice Hall, New Jersey, ch 3–5

- Kew N (2007) The apache modules book: application development with apache. Prentice Hall, New Jersey
- Lites B, Casini R, Garcia J, Socas-Navarro H (2007) A suite of community tools for spectro-polarimetric analysis. *Memorie della Societa Astronomica Italiana*, vol 78, pp 148
- Mathiassen L (2000) Object oriented analysis and design. Marko Publishing, pp 173–187
- Potter N, West P, Gallagher J, Garcia J, Fox P (2006) OPeNDAP Server4: Building a high-performance server for the DAP by leveraging existing software. *Eos Trans. AGU*, 87(52), Fall Meet. Suppl., Abstract IN13D-1181
- Quinn M (2004) Parallel programming in C with MPI and OPeMP. McGraw-Hill, New York, pp 219
- Szyperski C (1998) Component software, beyond object oriented programming. Addison-Wesley, Massachusetts, ch 4