A Performance Oriented Design Methodology for Large-Scale Distributed Data Intensive Information Systems

Daniel A. Menascé
menasce@cs.gmu.edu
Dept. of Computer Science
Center for the New Engineer

Hassan Gomaa
hgomaa@isse.gmu.edu

Larry Kerschberg
kersch@gmu.edu

Dept. of Information and Software Systems Engineering
Center for Information Systems Integration and Evolution
George Mason University
Fairfax, VA 22030-4444, USA

Abstract

The Earth Observing System (EOS) Data and Information System (EOSDIS) is perhaps one of the most important examples of a large-scale, geographically-distributed, and data-intensive systems. Designing such systems in a way that ensures that the resulting design will satisfy all functional and performance requirements is not a trivial task. This paper presents a performance-oriented methodology to design large-scale distributed data intensive information systems. The methodology is then applied to the design of the EOSDIS Core System (ECS). Performance results, based on queuing network models of ECS are also presented.

1 Introduction

One of the most important examples of information systems that are large-scale, geographically distributed, and handle very large volumes of data is the Earth Observing System (EOS) Data and Information System (EOSDIS). EOS is a NASA program mission to study the planet Earth. A series of satellites with scientific instruments aboard will be launched starting in 1997. They will send an estimated terabyte/day of raw data about the atmosphere, land, and ocean. George Mason University (GMU) was one of three (the others were UC Berkeley and North Dakota) selected universities to develop an independent architecture for EOSDIS Core System (ECS). GMU put together an interdisciplinary team composed of Earth scientists, computer, and information scientists. The Earth scientists of our team came from GMU's Computational Science and Informatics Institute, from the University of Delaware, the University of New Hampshire, and from the Center for Ocean-Land-Atmosphere Studies (COLA) in Maryland. The authors of this paper were involved with the computer and information aspects of the architectural design. A methodology had to be developed to design such a complex system. This methodology is performance oriented to ensure that the final design would satisfy the functional performance requirements of the system. The methodology is general and can be applied to the design of any large-scale distributed data intensive information system. After presenting the methodology, we discuss how it was applied to the design of ECS.

2 Large-scale distributed data intensive information systems

This section characterizes a large-scale distributed data intensive information system (LSS), provides the principles to be used when designing an LSS, and gives the logical architecture of an LSS building block (the LSS Node).

2.1 Characterization of an LSS

Figure 1 depicts the various components of a LSS. Such systems can be characterized as follows:

- *Large number of users:* the number of potential users of an LSS can range from tens of thousands to millions of users.
- *Diverse user population:* users may include researchers studying a particular domain of science running complex simulation models, policy makers at governmental agencies, international organizations, private industries, and K-12 students. Moreover, the users of an LSS are assumed to be spread over very large geographical areas.
- *Diversity in user requirements:* as a consequence of the diversity in user population one may also expect to have a wide variation in user requirements. While some users may pose very simple queries to the system, other users may submit complex requests that may involve evaluating very complex scientific models or correlating several image files. User requests may also vary widely in terms of the amount of data requested (from a few bytes to hundreds of gigabytes). Different types of users may have different performance requirements and different categories of users may be assigned different priorities to ensure that their performance requirements are met.
- *High data intensity:* raw data is expected to arrive at an LSS from one or more sources (e.g., instruments aboard Earth orbiting satellites, particle accelerators) at very high rates (e.g., from terabytes to petabytes (10^{15} bytes) per day).
- *Diversity in data types stored:* the holdings of an LSS are assumed to include a large variety of data

0-8186-7123-8/95 $4.00

types, such as raw data, large data sets resulting from applying complex algorithms to the raw data, images, metadata (i.e., data describing the data, instrument, and calibration), free format text, and multimedia documents.

- *Function distribution:* the different functions of an LSS should be implemented by components (LSS nodes) that are geographically distributed. These components are connected through one or more interconnected networks. Distribution is important to provide modularity, fault tolerance, and scalability to the design of an LSS.

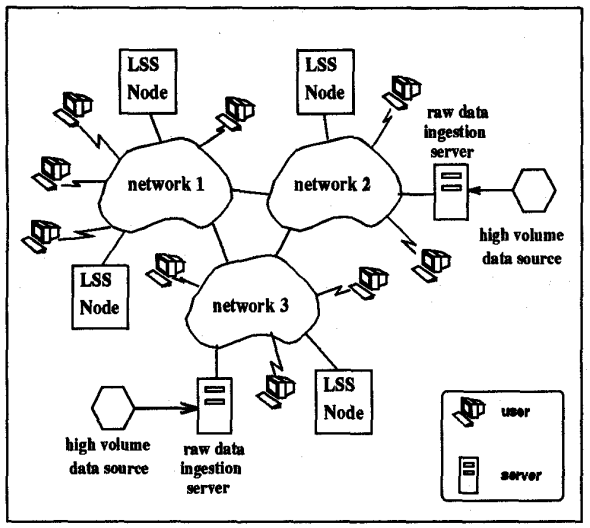

Figure 1: A large scale distributed data intensive information system

2.2 Design principles for an LSS

An LSS should adhere to the following principles:

- *P1. Location Transparency*: users should be able to access any information object, including data products, metadata, and browse data, without having to know the physical location of these objects. This implies that data can migrate to achieve load balance, cope with failures, and improve performance without disrupting the users of the system.
- *P2. Modularity*: the architecture of an LSS should be composed of elements that can be configured to serve as nodes of different type, different data processing capabilities, and different storage capabilities, with the same underlying architecture.
- *P3. Minimization of User Connections*: the number of users that will access an LSS is very large. Thus, the system should allow for users to do as much local work as possible before a connection to the LSS is made.
- *P4. Load Balancing:* the design should allow for the system load to be automatically balanced among the network of LSS nodes or even user computing facilities. An LSS should be capable of registering idle CPU cycles at user facilities and schedule computations using these cycles.
- *P5. Separation of Functions:* the functions provided by an LSS should be divided into related groups and implemented by separate types of servers. This way, servers can be optimized to perform the functions they are best suited for.
- *P6. Scalability:* an LSS should be scalable to take into account different requirements that may exist at different stages of its lifecycle. Requirements may change as new data sources are incorporated into the system (e.g., new data collection satellites being launched) and as new users learn about the system. The system should be scalable on a selective basis. This means that if more I/O capacity at a given site is required, it should be possible to upgrade the I/O subsystem without necessarily impacting the processing and networking capabilities.
- *P7. Support for Heterogeneity:* an LSS should support different types of processing paradigms to accommodate the diversity of processing needs that may occur within a given complex application or in the collection of applications submitted to the system. Machines of different processing paradigms as well as heterogeneous architectures should be allowed to coexist at the processing servers. The motivation for heterogeneity has been demonstrated in [13]. The scheduler should consider the heterogeneity when making its decisions. Significant work has been done in the area of metacomputing [3] and on scheduling of parallel applications in heterogeneous environments [10, 12].
- *P8. Minimization of Data Transmission:* Data transmission should be kept to the minimum possible level. This implies that data sets should be transmitted from source to destination with the minimum possible number of intermediate nodes. This principle is particularly important in a LSS where huge volumes of data are transported in and out of the system. A virtual client protocol was proposed [8] for minimizing data transfers in nested client/server interactions.

2.3 Logical architecture of an LSS

The basic building block of an LSS is the LSS node shown in Fig. 2. The functions of an LSS are provided by a collection of interconnected LSS nodes.

The basic logical architecture of such a node includes a collection of servers that implement the different functions of an LSS. Servers in an LSS node may act as clients with respect to other servers in the same or other LSS node. The main types of servers in an LSS node are:

1. Archival server: handles the storage of all types of data in an LSS node. This type of server may be further specialized into archival servers of different types.
2. Metadata server: manages the collection of metadata relative to the data managed by the archival server.

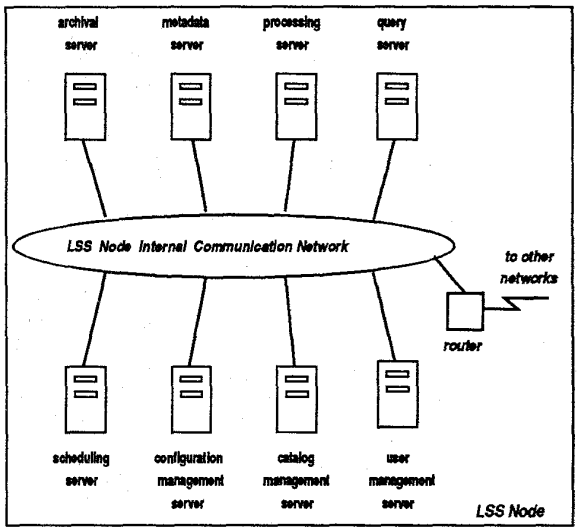

Figure 2: Logical architecture of an LSS node

3. Processing server: handles processing requests to transform data sets of one type into data sets of another type.
4. Query server: manages the processing of both ad-hoc and pre-registered queries.
5. Scheduling server: schedules the processing requests using both local and remote processing servers. The set of scheduling servers in all LSS nodes, collectively implement a global scheduler.
6. Configuration management server: monitors the operation conditions of the LSS node, collects statistics about the utilization of its various resources, and reconfigures the node when necessary to cope with failures and performance degradation.
7. Catalog management server: maintains a directory of all objects managed by the LSS. The collection of all catalog managers collectively maintain a global directory of LSS objects. The catalog managers are used to locate LSS objects.
8. User management server: maintains information about registered users, their profiles, accounting and security information.

3 The design methodology

Performance models play a crucial role in the design of complex information systems. They are useful to distinguish among a variety of alternatives, both good and bad, assess the impact of architectural choices, predict potential bottlenecks, size hardware components, and evaluate if a proposed architecture will meet the performance requirements under the expected workload.

This section describes a performance oriented system design methodology. The main thrust of this methodology is to ensure, by successive refinements, that the architecture meets performance goals set forth in the requirements analysis and specification phase. The methodology is better explained with the help of Figure 3.

There are three basic inputs to the methodology: functional requirements, performance requirements, and the user model. These three elements are shown as shaded clouds in Fig. 3. The functional requirements specify the functions to be performed by the system. The performance requirements specify the requirements on performance when executing any of these functions (e.g., maximum response time values, minimum throughputs). The user model describes the typical interactions between users and the system. The user model also provides quantitative information on the frequency with which users interact with the system, as well as the resource requirements per interaction (e.g., an Earth scientists studying ocean circulation models will typically browse twenty 3 MByte images and then will run an ocean circulation model that requires an average of 500 MFLOPs).

A domain model [4] is developed that reflects, at the functional level, the interaction of the main system components. The resulting domain model is then used to derive a client/server software architecture specification which depicts the message exchanges between clients and servers in the system. This client/server architecture is combined with the user model to generate event sequence scenarios showing each type of user interaction. These scenarios are further annotated with performance parameters such as request arrival rates, data volumes per request, server processing and I/O requirements per request. The client/server software architecture drives a first-cut at the system architecture. The client/server software architecture and the system architecture generate a software/hardware mapping that associates logical servers to physical elements such as processors and network segments. The components of the system architecture are assigned performance characteristics (e.g., network segment speeds, router latencies, I/O subsystem bandwidth, processor speeds). Then, the performance annotated scenarios, the software/hardware map, and the architecture performance characteristics are combined to generate input parameters to a performance model. The performance model is based on analytical methods to solve mixed (i.e., open/closed) queuing networks [9]. The outputs of the performance model include response times and throughputs for each type of request submitted to the system. An analysis of the results of the performance model reveals the possible bottlenecks. If the architecture does not meet the performance objectives, architectural changes at the hardware and/or software level have to take place. These changes are guided by the outputs of the performance model. These changes are reflected back into the architecture and into the event sequence scenarios. Successive iterations ensure that the final design meets the performance objectives. Since the design process is iterative, one starts with a first cut at the architecture and goes through successive refinements to meet the performance goals. These refinements may imply combining servers into a single physical computing element, changing the software architecture by creating additional servers, or changing the underlying hardware characteristics (e.g., internal network bandwidth, processing element speeds, and I/O subsystem rates).

The methodology just described was used by the authors in the design of an alternative architecture for EOSDIS Core System (ECS). The next section briefly de-

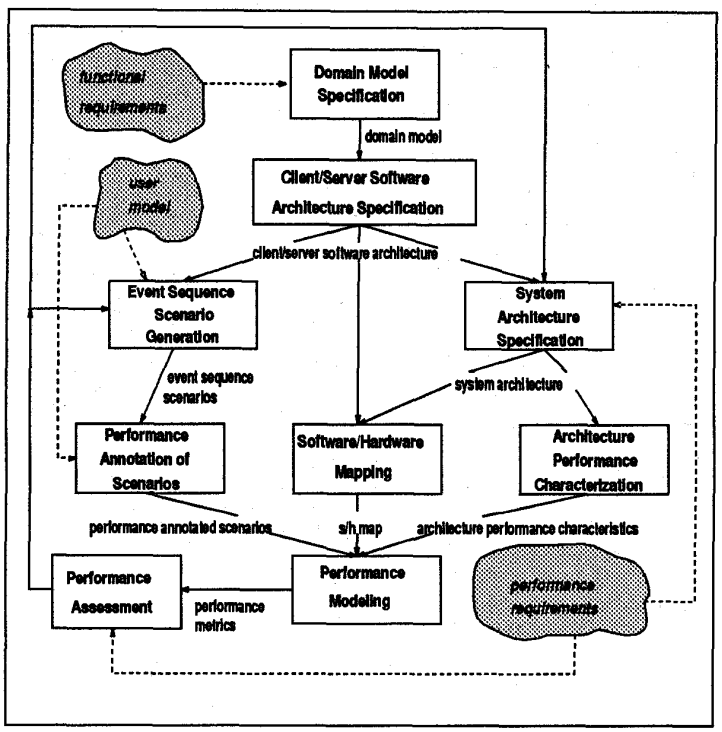

Figure 3: Performance oriented design methodology

scribes EOSDIS and ECS. The remaining sections of the paper discuss how the methodology was applied to the specific design.

4 EOSDIS and ECS

Raw data coming from the NASA satellites is first received at the White Sands complex in West Virginia. After some initial level of calibration, it is sent for archival and further processing at a collection of eight centers called Distributed Active Archive Centers (DAACs). The raw data received by the DAACs is called Level 0 data. Level 0 data is used to generate Level 1 data, defined as reconstructed, unprocessed instrument data at full resolution, time-referenced, and annotated with ancillary info. Environmental variables at the same resolution and location as the Level 1 data are derived to generate Level 2 data. A set of variables mapped onto uniform space-time grid scales, with some consistency and completeness, are called Level 3 data. Finally, the model output or results from analyses of lower level data is referred toas Level 4.

About 500 NASA selected scientists will determine which standard data products are to be generated by ECS. The facilities where these scientists are located are called SCFs (Science Computing Facilities).

The rest of this paper shows how the methodology was applied to the design of ECS. Many details had to be ommitted due to space limitations.

5 ECS node logical architecture

The building block of the ECS architecture is called an *ECS node*, which is a collection of servers of different types. This is ECS's specialization of an LSS node. ECS nodes can be configured to serve as DAACs, Auxiliary Data Centers, or even SCFs. ECS nodes are connected to the user community by a User Network (e.g. Internet/NII) and are connected to other ECS nodes by the ESN network.

An ECS node (see Fig. 4) has three main subsystems: ECS Object Management Subsystem (EOMS), Product Processing Subsystem (PPS), and Information Management Subsystem (IMS). Each of the main subsystems is implemented by a collection of servers to be described below. All servers can communicate with one another through an ECS Node local high speed network. Through this network, and ECS node is connected to the Internet and to the Earth Science Network (ESN). Note that Fig. 4 represents a logical not a physical design. In order to map the logical design into a physical design one would need to specify the mapping of the various servers into actual machines, the capabilities of these machines in terms of processing and I/O characteristics, and the topology and bandwidth characteristics of the ECS Node Local high speed network. More details on the specific functions of each server are given in [8].

An ECS node may have any number of servers of each type. The actual number of servers is determined by the workload imposed on a given ECS node. A server may act as client for a server on the same or on a remote

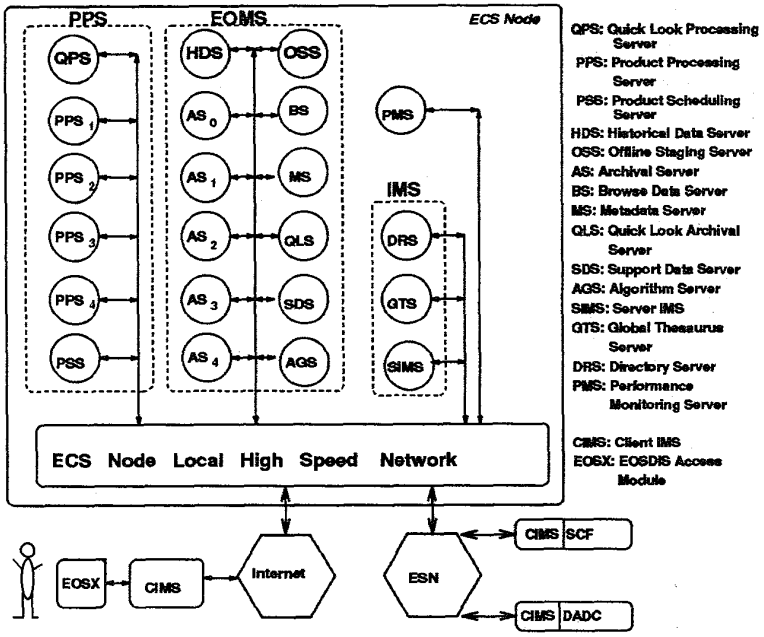

Figure 4: ECS node architecture.

ECS Node. For example a Product Processing Server, acting as a client, may request a data product from several Archival Servers. Some of them could be local to the Product Processing Server and others could be remote. It is important to realize that servers are just logical concepts. The mapping between servers and actual machines determines the actual physical architecture of the system.

Although it is conceivable for all servers of an ECS node to be mapped into a single computer, this is not desirable for various reasons: the need for redundancy, adequate performance, and the need to accommodate significantly different processing, communications, and I/O requirements.

6 A performance model for ECS

Performance models of computer systems are used to predict how performance metrics such as throughput, response times, queue sizes, and component utilizations, vary as a function of the workload and system parameters. The analysis discussed in this section is based on queuing network based analytic models [9]. We present here a brief overview of the concepts behind queuing network (QN) models as well as the terminology to be used.

A *queuing network (QN)* is a network of queues through which customers flow. Customers may have different meanings in different contexts. In the EOSDIS context, a customer may be a request to retrieve a browse image from a certain DAAC, or a request to generate a given standard product. A *queue* is composed of a *service center* and a waiting line of customers waiting to use the service center. Service centers may be used to represent a processing element, components of an I/O subsystem, or a communications network. Since customers may vary significantly in terms of the demands placed on the different service centers, one should aggregate all similar customers into groups called *customer classes*. All customers of the same class are represented by a set of values that represent the average demand on each service center over all customers of the class. In the context of EOSDIS, each different scenario generated from the user model may give rise to a different customer class. Some customer classes may be considered to be *open* in the sense that the customers arrive from outside the system, get served by a subset of the service centers, and leave. In this case, there is not limit on the number of customers in the queuing network. An example of an open class could be "browse image queries from K-12 students". Other classes may be considered to be *closed* in the sense that there is always a constant number of customers of this class in the queuing network. This type of class is used to represent work that is performed on a routinely and continuous basis. An example of this would be "processing of standard product MOD28".

Each customer class is specified by the following parameters:

- type: open or closed.
- intensity parameters: arrival rate of customers in the case of open classes and number of customers in the case of closed classes.
- service demands: total time spent per customer receiving service from each service center. Note that the time spent waiting to get access to the service center is not part of the service demand.

The total time spent by a customer in the queuing network is the response time for the customers' class. The response time is composed of two basic components: queuing time and service time. The queuing time, computed by queuing network based analytic models, is a

function of the contention for access to the several service centers. The service time is a function of the total service demand placed on all resources. Another performance measure of interest is the throughput per customer class. This measures the average number of customer request completions per unit time. For open classes, the throughput is simply equal to the arrival rate. For closed classes, the throughput is computed by a queuing network based analytic model.

For each customer class, one must specify the required *service levels*, i.e., the upper and lower bounds on performance. Examples of service levels in the context of EOSDIS are:

- level 1 data must be made available within 48 hours of observation.
- levels 2 and 3 standard products should be made available within 96 hours of observation.
- A DAAC should be capable of generating quick-look products within 1 hour of receipt of necessary input for 1% of EOS instrument data.

Each user category may exhibit several important patterns of interaction with ECS. Each scenario is assigned to a class of customers in the queuing network model. Each scenario is mapped to the ECS architecture so that service demands can be obtained for the different architecture components and subsystems. Given that the architecture under consideration is a client-server based architecture, multiple time-line diagrams such as the one shown in Fig. 5 were used to map each scenario to the architecture. In this figure, an external request submitted to the Client IMS implies in three requests sent to Server IMS, one request sent to the Level 2 Metadata Server, and one request sent to the Level 2 Browse Data Server. Note that servers may act as clients when requesting service from other servers. The dashed arrows from a server (acting as a client) into a server indicate a request. These arrows are annotated with performance related data such as the probability that service is requested from this server, number of bytes involved in the request, and resource demand parameters related to the service requested from the server. The dashed arrows from a server to another (acting as a client) indicate replies from previous requests. These arrows are also annotated with the number of bytes sent back to the client.

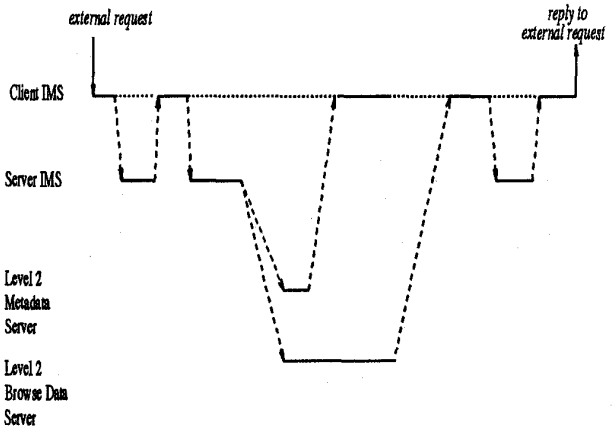

Figure 5: Multiple time-Line diagram

An analysis of each scenario determines the average number of requests per server as well as the average service demand per server. This analysis also determines the average load imposed on the various communication subsystems of ECS.

6.1 Performance model parameters

The service demand parameters per server for each scenario (customer class in the queuing network model) are obtained as follows. Let

- $b(r)$: data volume to be retrieved by request r (in MBytes),
- $c(r)$: computational demand associated with request r (in millions of floating point operations),
- C_i: computing speed of server i (in MFLOPS),
- IO_i: I/O bandwidth of server i (in MBytes/sec),
- $p_{i,j}(r)$: probability that request r is addressed to server j,
- $\mathcal{R}_i$: set of requests generated by client i,
- D_j^{proc}: processing service demand at server j,
- D_j^{IO}: I/O service demand at server j.

Then we can write that,

$$D_j^{\text{proc}} = \sum_{\forall i} \sum_{r \in \mathcal{R}_i} \frac{p_{i,j}(r) \times c(r)}{C_i}$$

$$D_j^{\text{IO}} = \sum_{\forall i} \sum_{r \in \mathcal{R}_i} \frac{p_{i,j}(r) \times b(r)}{IO_i}$$

6.2 The queuing network model

The performance model used for the ECS architecture is a mixed queuing network: some classes are open and some are closed. The data product generation classes are closed classes while queries (e.g., browse and metadata) are represented as open classes. Any server (in the client/server architecture) is represented by two devices in the QN: a computing device and an I/O device. The various communication networks are represented by load dependent devices in the QN. Figure 6 contains a diagram of the queuing network model used to represent the ECS architecture.

6.3 Performance results

Several numerical examples are used in this section to evaluate the architecture of ECS. A baseline model using the values derived from the user model generated by the team of Earth scientists was evaluated first. Modifications analysis were then carried out to gauge the sensitivity of the architecture to different changes in the workload and architecture components. It should be noted

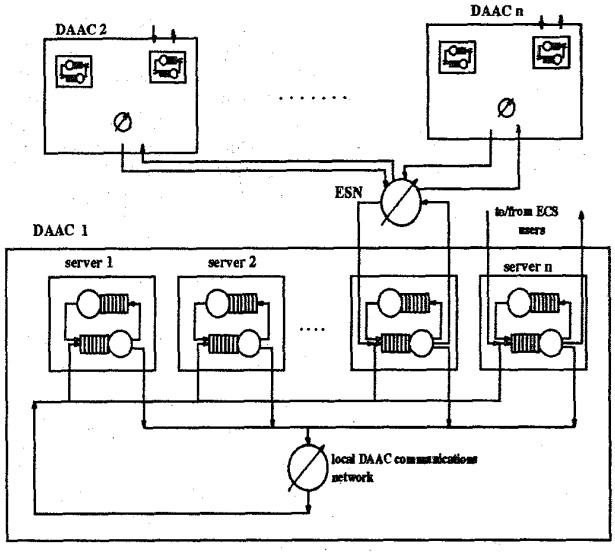

Figure 6: Queuing network model of ECS

that the results shown below do not represent actual ECS performance since, at the time of this study, we did not have enough data available. Some assumptions had to be made to compensate for missing values. These assumptions are likely to change. At any rate, the results presented here indicate the type of analyses that can be made with the performance models described in the paper.

The following numerical data are considered in the baseline model:

- Local Area Network Bandwidth: 100 Mbps based on a 2-channel FDDI backbone.
- ESN bandwidth: 45 Mbps assuming T3 lines between the major DAACs.
- Processing capacity: 60 GFLOPS, based on the achieved performance for a CM-5 running the LAPACK benchmark [1].
- I/O bandwidth: 70 MBytes/sec.

For the purposes of this study we considered the following scenarios:

- *El-Nino and Southern Oscillations (ENSO)* [5].
- *World Ocean Circulation Experiment and the Tropical Ocean Global Atmosphere (WOCE/TOGA)* [6].
- *Global Ocean Observing System (GOOS)* [6].
- *Terrestrial Scenario (Land_use)* [7].
- *Push scenario for Level 0 products (L0).* This scenario represents the workload imposed on ECS by the continued arrival, processing, and storage of level 0 products. The numerical data for this workload is derived from tables available in HAIS documents.
- *Push scenario for Level 1 and higher data products (L1-L4).* This scenario represents the workload imposed on ECS by the processing and storage of levels 1 through 4 data products. The numerical data for this workload is derived from tables available in HAIS documents.

We show here two examples of the results obtained in our analysis. An extensive set of curves and tables can be found in [11]. The value of the workload intensity for the user scenarios which are not varying were chosen so that they represent a light load. For the open scenarios (ENSO, WOCE/TOGA, GOOS, and Land_use) the arrival rate was fixed at 0.1 requests/sec except when this is the varying parameter. For the L0 and L1-L4 scenarios the number of jobs in the system was fixed at 10 except when this is the varying parameter.

Figure 7 displays the impact of varying the arrival rate of ENSO requests on the response time of GOOS, Land_use, and L1-L4 classes. As it can be seen, response

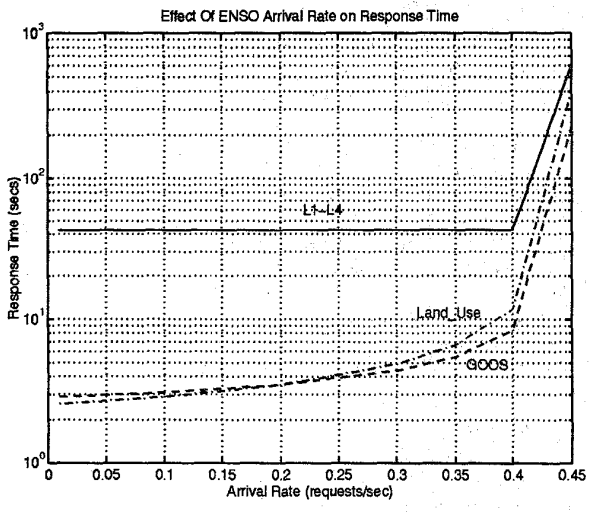

Figure 7: Impact of ENSO arrival rate on GOOS, L1-L4, and Land_Use scenarios.

times smaller than 4 sec for Land_Use and GOOS are supported for arrival rates of ENSO requests not exceeding 0.25 req/sec. After 0.4 req/sec for ENSO requests, the system saturates and the response times increases at a very fast rate. The L1-L4 workload is rather insensitive to ENSO requests until the onset of saturation. Figure 8 investigates the impact of consolidating some of the DAACs. In particular, the loads of DAACs MSFC, JPL, UAF, and CU are assumed to be assigned to LaRC. This reduces the number of DAACs from 8 to just 5. The vertical axis in the figure is the response time ratio S defined as

$$S = \frac{\text{Response Time Under the Consolidated Scenario}}{\text{Response Time Under the Original Scenario}}$$

A value of S greater than 1 indicates that consolidating DAACs increases the response time. It should be noted that when DAACs are consolidated, there are two effects on performance: some servers will become more heavily utilized as a result of the additional load. This makes S to be greater than 1. On the other hand, for some

scenarios, the network demand is reduced by the DAAC consolidation. This reduces the response time under the consolidated scenario and makes $S < 1$. As seen in Fig. 8, performance for the ENSO workload becomes twice as bad when the arrival rate of GOOS requests approaches 0.55 req/sec. The L0 workload is only slightly sensitive to DAAC consolidation.

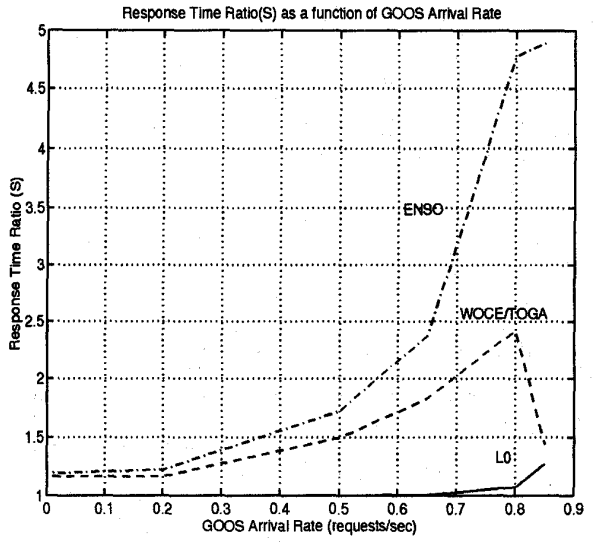

Figure 8: Impact of GOOS arrival rate on ENSO, L0, and WOCE/TOGA scenarios.

7 Conclusions

A characterization of large-scale, distributed, and data intensive information systems was given. A general methodology to design such systems was discussed. The methodology is performance-oriented. Through the iterative use of performance models to carry out performance prediction, the system design is refined through successive steps. This ensures that the final design will satisfy the functional and performance requirements. EOSDIS Core System was used as a case study to illustrate the application of the methodology. The performance results are preliminary at this stage.

Acknowledgements

This work was partially supported by Hughes Applied Information Systems (contract ECS-00010). The authors would like to acknowledge the many useful discussions they had with the ECS Independent Architecture Study Group at GMU led by Menas Kafatos. In particular, they would like to thank Jim Churgin, Ferris Webster, Berrien Moore III, and Jim Kinter, for explaining them the different aspects of Earth science and scientist requirements for EOSDIS. They would also like to thank Sudhanshu Killedar for writing the programs that implement the analytic model.

References

[1] Dongarra, J., H-W. Meuer, and E. Strohmaier, TOP 500 Report 1993.

[2] NASA, EOS Reference Handbook, Washington, DC, August 1993.

[3] Freund, R. F. Optimal selection theory for superconcurrency. *Proc. Supercomputing'89*, IEEE Computer Society/ACM Sigarch, Reno, NV. 1989, pp. 699-703.

[4] Gomaa, H., L. Kerschberg, and V. Sugumaran, *A Knowledge-Based Approach for Generating Target System Specifications from a Domain Model* , Proc. IFIP World Computer Congress, Madrid, Spain, September 1992.

[5] Kinter, J., B. Doty, and J. Shukla, *Meteorological Scenarios*, The GMU ECS Federated Client-Server Architecture, Final Report, Chapter 5, Part II, George Mason University, August 31, 1994.

[6] Churgin, J. and F. Webster, *Oceanographic Scenarios*, The GMU ECS Federated Client-Server Architecture, Final Report, Chapter 4, Part II, George Mason University, August 31, 1994.

[7] Moore III, Berrien, *Terrestrial Scenarios*, The GMU ECS Federated Client-Server Architecture, Final Report, Chapter 3, Part II, George Mason University, August 31, 1994.

[8] Menascé, D. A., L. Kerschberg, and H. Gomaa, *ECS Client-Server Systems Architecture*, The GMU ECS Federated Client-Server Architecture, Final Report, Chapter 2, Part III, George Mason University, August 31, 1994.

[9] Menascé, D. , V. Almeida, and L. Dowdy, *Capacity Planning and Performance Modeling: from mainframes to client-server systems*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[10] Menascé, D., S. Porto, and S. Tripathi, *Static Heuristic Processor Assignment in Heterogeneous Multiprocessors*, International Journal of High Speed Computing, Vol. 6, No. 1 (March 1994).

[11] Menascé, D. and S. Killedar, *Performance Modeling of ECS*, The GMU ECS Federated Client-Server Architecture, Final Report, Chapter 5, Part III, George Mason University, August 31, 1994.

[12] Menascé, D.A., D. Saha, S. Porto, V. Almeida, and S. Tripathi, *Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures*, J. of Parallel and Distributed Computing, V. 28, No. 1, July 1995.

[13] Menascé, D. and V. Almeida, *Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures*, Proc. of the ACM-IEEE Supercomputing'90 Conference, New York, NY, USA, Nov. 12-16, 1990.