UCRL-TR-211040

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

MObIUS (Massive Object Integrated Universal Store): A Survey Toward a More General Framework

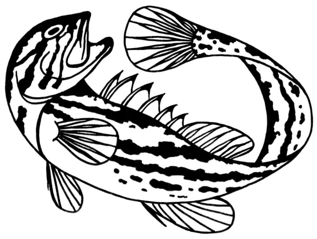
J. K. Sirp, S. T. Brugger

April 5, 2005

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

M O b I U S

(*Massive Object Integrated Universal Store*)



A Survey Toward a More General Framework

Jennifer K. Sirp
and
S Terry Brugger

February 23, 2004

Abstract

General frameworks for distributed computing are slowly evolving out of Grid, Peer Architecture, and Web Services. The following results from a summer long survey into distributing computing practices have revealed three things. One, that Legion and Cactus-G have achieved the most in terms of providing an all-purpose application environment. Two, that extending a local programming environment to operate in a highly distributed fashion can be facilitated with toolkits like Globus. Three, that building a new system from the ground up could be realized, in part, by using some of the following components; an Object Oriented Database, Tapestry, JXTA, BOINC, Globus, component architecture technology, XML and related libraries, Condor-G, Proteus, and ParMETIS.

Introduction

This paper offers the results obtained after conducting a broad survey of computer systems. The objective of this research was to locate systems that could potentially contribute to the development of MObIUS, a general, distributed, Object-Oriented, framework. In contrast to current implementations of Web Services, Peer-to-Peer and computational Grid middleware, this system should be able to efficiently handle a wide variety of applications and adapt to a dynamic environment. Many systems were evaluated over the summer. Those showing the most promise are featured here. The requirements that were used as a means of measurement and evaluation throughout this study are discussed on the page that follows.

Requirements

1. Scalabity:

The proposed application should be highly scalable, ranging from a single machine to a nation-wide grid or peer environment. The system should have the ability to make full use of the resources that are currently available.

2. Application Development:

The framework should allow the application developer to compose, deploy, and manage applications while operating at a high level of abstraction.

3. Interoperability:

An application developer should be able to define and compose new objects on new nodes without the need to worry about binary compatibility.

4. Granularity:

Data modeling should be achieved through object-level granularity. Ultimately, this will allow concepts such as versioning, capabilities, distribution, partitioning, performance, and migration to be applied to any object regardless of size or complexity.

5. Access:

An object level security mechanism will be used in this system, providing security from the ground up. Access to all objects should be achieved through capabilities. A capability acts as a reference to an object, defining both a name and access level. An object may have any number of capabilities.

6. Distribution and Migration:

Objects should be initially distributed across the available store (primary and secondary, which may be on different nodes). The system should also impose object migration based on access patterns and resource availability (i.e., load balancing and distributed computation).

7. Performance:

Replication of (caching) objects should occur based on access patterns. Furthermore, the system should allow a user to specify the minimum level of replication *of an object for robustness. Optimally, the system will perform automatic partitioning by splitting and distributing large objects across nodes for efficiency.*

8. Versioning:

Versioning should be provided for objects and allow multiple branches to occur. Users should be able to disable versioning for efficiency. Users should also be able to own different versions or branches. Replicants should be updated upon user request or automatically by the system. Replicants should have the ability to be read-only, write through, or versioned writes with conflict resolution.

Organization

This paper is organized into three parts. The first section briefly describes the current state of distributed computing to establish a background for why we feel this project is important. The second section explores systems that have achieved some aspect of what we hope to accomplish and discusses what these systems can contribute to the project. Section three contains comparison charts which contrast the differences between various systems.

1. Background

At the moment, distributed computing has manifested itself in three distinguished realms, namely, Grid, Peer-to-Peer, and Web Services. What is interesting is that while each of these technologies differs based on the interests of the people behind them, they are all moving toward the same goal: finding a more general, application framework [1]. There is a definite need for a general application environment that can make working with complex aggregations, like Grids and Peer groups, easier for the developer. It is in the intersection of Grid, Peer, and Web technologies where it seems most reasonable that an effective middleware solution lies.

Computational Grids have had great success from a high performance and scientific standpoint. However, their success has been attained from harnessing the power of a few enormous machines rather than a large collection of less powerful computers [1]. As a result, their middleware hasn't been designed with the need to operate on a highly dynamic bed of resources. Furthermore, the scientific community has been focused on optimizing performance for specific computation problems. These individual projects have been focusing their energy into either gathering resources, in terms of Virtual Organizations, with a variety of security and interests issues, and/or fine tuning algorithms for the Grid environment. As a consequence, Grids are difficult to program. The application developer has to work at an undesirable level of abstraction, paying attention to a variety of details about the architecture beneath [1].

Web Services have had great success from the client/server perspective of distributed computing[a]. Currently, Web Services have achieved the ability to integrate homogeneous distributed systems that are running the same client/server environment [2], and seem to be doing well in heterogeneous environments, for example, when mixing Windows and Unix servers. Web Services have been able to provide an easy development environment for the programmer with .NET and J2EE, although they are somewhat limited as to the types of services that they can provide. The solution to their interoperability problems has been light-weight, flexible middleware, namely SOAP (Simple Object Access Protocol) [2]. Web Services are based on a simple messaging protocol to ease application development and provide interoperability among disparate systems. UDDI (Universal Discovery, Description, and Integration) and WSDL (Web Services Description Language) handle the dynamic discovery and composition of these services[b].

Peer-to-Peer technology is another distributed computing paradigm that has demonstrated promise in the hopes of providing a ubiquitous framework for applications. Peer technology has been unmatched in regards to its degree of decentralization and ability to adapt to unstable resources [1]. Peer architecture has been designed with the need to be highly self-organized. However, the peer community is also fragmented in that peer groups are designed around specific purposes: instant messaging, file sharing, and single purpose programs. As it stands, users are required to download multiple client/server programs in order to obtain a specific service. Furthermore, Peer-to-Peer architectures do not take data storage and application processing with inter-peer dependencies into account [1].

The focus of our research has been aimed at the aforementioned technologies. In general, by combining the best aspects of each, a framework could be achieved that would provide a solution to the difficulties in distributed computing. Combining the flexible protocols used in Web Services, the self-organization of Peers, and the high performance development environment of Grid architecture, a versatile and scalable system could easily evolve. The proposed middleware would be able to flex with the dynamic nature of the Internet and be able to take full advantage available resources.

---

[a] For clarification, the term Web Services is used here to describe client/server applications. In a typical Web Services scenario, a client application sends a request to a service provided at a given URL using the SOAP protocol (XML over HTTP). The service receives the request, processes it, and returns a response to the client.

[b] UDDI (Universal Discovery, Description, and Integration) is a specification for a registry that is used to publish WSDL documents. This registry combined with WSIL (Web Services Inspection Language) is used to locate services. Web Services Description Language (WSDL) is a standard interface for Web Services. WSDL is defined by an XML schema. These documents allow multiple protocols to be associated with each service.

2. Systems Overview

The following section presents and discusses the various systems that were evaluated. The most promising of these are Legion and Cactus-G. Both of these systems encompass a majority of the proposed requirements. Furthermore, these systems are based on a component or object model, lending them to being easily extendable. In case these systems are not applicable, other components were surveyed as potential contributors MObIUS. These subsystems are described and discussed below.

2.1 System Extension

Four systems were evaluated in this category. Two of these, Legion and Cactus-G, are the best representatives of complete systems that address a majority of the requirements. Both systems follow an Object-Oriented design making them feasible candidates for extension. Amoeba and NLTSS (Network Livermore Time Sharing System) were also examined, however, NLTSS has been retired and Amoeba appears unstable. For a detailed comparison of all four systems, see Appendix A, Table 2.1.

Legion is a distributed Object-Oriented middleware system that runs on the operating system of a variety of machines. Systems that run Legion, contribute resources into a collective pool that is then presented to the user as if it were a single machine. Legion was built on top of Mentat, a very stable and efficient parallel processing development environment[c]. As a result, high performance, interoperability, and application execution are automatically achieved. All objects are Legion objects and are distributed over the available resources. The object model enables developers to create and reuse application code easily. An IDL (Interface Description Language) is used to provide interoperability among the objects over remote locations. Unlike CORBA (Common Object Request Broker Architecture), Legion developed its object communication techniques with high performance in mind, making it better suited for parallel programming over a wide-area network. Legion is implemented in C++ and follows an object design model, making it a likely candidate for extension.

Another potential candidate for extension is Cactus-G. Cactus began as a HPC (High Performance Computing) scientific development environment that was later extended to run on Grid architecture. MPICH-G2, a Grid enabled implementation of

[c] Particularly well-suited for message passing, non-shared memory architectures, Mentat is an Object-Oriented parallel processing system. Objectives of Mentat include; easy-to-use parallelism, high performance through parallel execution, and interoperability. Mentat utilizes the Object-Oriented paradigm to provide high-level abstractions for the complex aspects of parallel programming, such as communication, synchronization, and scheduling. The run-time system constructs program graphs, manages communication and synchronization, and is portable across a wide variety of MIMD architectures. Mentat has been distributed to over 100 different sites and is freely available.

MPI[d] (Message Passing Interface), provides the communication protocol over a wide area network and Globus is used for authentication services. Cactus was originally developed as a framework for the numerical solution of Einstein's equations, and has now evolved into a more general problem-solving environment. This new framework allows developers to work at a high level of abstraction from the complex Grid architecture underneath. Cactus runs on the user's operating system like Legion and features a microkernel design, which enables 'thorns' to be added separately. Different modules, or thorns, can implement different versions of an application, solve completely different problems, or provide general services such as data migration and replication. The code is freely available and the project representatives are very willing to assist and answer questions. The source code for Cactus is available through CVS and compiles easily; however, it will not actually do anything unless thorns are added. Thorns provide the applications and infrastructure [4]. The modular design also allows programmers to stack modules on top of one another and switch out components based on the desired implementation. The Thorn Library is freely available and currently offers data distribution, migration, parallel I/O, and check pointing. Object versioning has not been addressed but could be added as a thorn. Thorns can be written in Fortran, C or C++.

Neither Cactus-G nor Legion directly offers object versioning, and provisions for automatic data partitioning at run-time are questionable. However, it seems reasonable that a Legion object could be extended and versioning capabilities could be added. Furthermore, it may be possible to address run-time partitioning with another application like ParMETIS. For Cactus-G, it may be possible to implement thorns that are capable of versioning and partitioning and add them as separate modules.

Another possibility is to use Globus to extend a local system, since it can manage many of the underlying details associated with distributed computing. A current trend is for organizations to take their existing applications and deploy them over a Grid for better performance. Academic projects such as Cactus-G and Condor-G were once local systems that have now been extended with the Globus Toolkit.

Developed at Argonne National Laboratory, the Globus Toolkit is an open source, OGSI (Open Grid Services Infrastructure)[e] compliant implementation of libraries and tools designed to get an application 'grid-enabled' [5, 6]. The toolkit provides management, security, discovery, and monitoring of Grid resources, all from a standard

[d] MPICH-G2 is a Grid enabled version of MPI 1.1 that has used the Globus toolkit to operate on a highly distributed environment. MPICH-G2 uses TCP for inter-machine messaging and MPI when applicable.

[e] OGSI (Open Grid Services Infrastructure). The OGSI refers to Grids that are based on a Web Services model. OGSI uses WSDL to define accessible Grid services that are located across the Internet via SOAP (XML/HTTP).

Java API. Version 3 of the toolkit specifies a container architecture in which Grid services can be placed to run in a specified hosting environment. The container architecture allows services to be developed without having to worry about the underlying protocols and transport bindings. Services publish their existence through WSDL (Web Services Description Language). Another benefit of Globus is that it provides excellent support for organizations that are implementing distributed applications.

2.2 Component Systems

Should it be necessary to design and build a system from the ground-up, the following section summarizes and discusses systems that could be of use. Relevant components to this project have been identified by the following categories: Peer Technology, Component Architecture, File Systems, Messaging and Communication, and Optimizations. The most applicable examples of each system category have been focused on in detail.

2.2.1 Peer-to-Peer Technology

Peers have naturally evolved into being highly self-organized in order to adapt to a dynamic environment[f]. In this regard, borrowing from peer technology could be beneficial. The decentralized nature of these systems allows them to be autonomous and tolerant to a highly dynamic environment, such as the Internet. Furthermore, advanced messaging algorithms and searching mechanisms used by third generation Peers are far more efficient than earlier approaches which used broadcast with TTL (Time To Live). The most relevant Peer applications to MObIUS are Tapestry, JXTA, and BOINC.

Tapestry is a third generation, overlay, Peer-to-Peer infrastructure that is continually being developed at the University of Berkeley. Tapestry offers high-performance, reliability, efficient messaging, adaptive algorithms, and the guarantee of file delivery in an operating network. Instead of using the broadcast and flood approach that can bog down the Internet, Tapestry uses a distributed hashing algorithm to efficiently message nodes and guarantee delivery[g]. Tapestry also utilizes 'hot spot caching' where it publishes pointers to cached documents that are frequently accessed to boost performance [7]. An implementation of Tapestry has been coded in Java and features an extensible API. The Java implementation also enables the application to be highly portable since it can run on any machine with a JVM. Furthermore, Ocean Store has chosen to implement on top of the Tapestry infrastructure. Though just released

[f] The dynamic environment here refers to unreliable IP addresses and the disconnectivity of resources.

[g] Earlier Peer ancestors, like Gnutella, have been unable to guarantee files in an operating network. Implementations of Gnutella usually keep the TTL (Time To Live) of queries fixed. Resources beyond the specified number of hops are unreachable.

(October, 2003), Ocean Store is a global storage application that is designed to offer what typical federated file systems have achieved, on an unreliable bed of storage resources such as the Internet. Nodes running Tapestry would be able to use Ocean Store as a global storage mechanism.

Another approach to borrowing from Peer architecture is to use a toolkit like JXTA (Juxtapose) or BOINC (Berkeley Open Infrastructure for Network Computing) and build a specialized Peer application. Both JXTA and BOINC are in the early stages of development and have growing communities of contributing developers. BOINC provides a C++ API and has just released their toolkit (September, 2003). Alternatively, JXTA provides a Java API and is also now available for download.

2.2.2 Component Architecture

Component or object-based architectures such as CCA (Common Component Architecture) and CORBA (Common Object Request Broker Architecture) offer an Object-Oriented approach to distributed computing. These models propose standards that describe what an object should look like and how the corresponding framework that handles them should operate. Components provide an easy way for a programmer to develop, compose, and manipulate applications. Applications can be created with existing components regardless of what language or architecture they were initially implemented. Component models can enable complex applications to be managed by scripts, and legacy applications can often times be easily wrapped in order to run on the new infrastructure.

CORBA can provide interoperability between different languages and platforms and locations via an IDL and can be implemented with Grid technology. CORBA operates by using an IDL communication mechanism providing transparency and interoperability. Object references persist, but are not utilized as capabilities. A Client makes a request through an object reference to a distributed object which is typed by an interface. The ORB (Object Request Broker) delivers the client request and returns the result.

CCA is a specification developed by the DOE that describes how to build portable software components that can then be reused in any CCA framework. The CCA has been focused on building applications and components for massively parallel computers. CCA supports Java, C, Python, C++ and Fortran. CCA ports are synchronized and designed for direct transmission between the components. Three implementations of this architecture are Ccaffeine (SPMD), Uintah (Threaded), and XCAT (Distributed).

Although no detailed comparison was drawn between the two technologies, CORBA is clearly a much larger solution that has had more time to evolve, and many free implementations of the standard exist.[h] CORBA offers bindings for C++, Java, Ada, Smalltalk and others. CCA offers fewer features and is relatively new, but is likely to be more reliable. Two advantages that CCA has over CORBA are that it was developed with high performance computing in mind and it is simpler to learn.

Component architecture could help achieve interoperability and provide a high level of abstraction to the programmer. Businesses that offer Web Services have been using CORBA to achieve a very high degree of interoperability among different machines and environments. This kind of technology has also been directly applied to science. XCAT, in particular, has implemented a Web Services model[i], using CCA, on a Grid environment. Multiple application components can be linked together and managed through the engine, to achieve a particular task. Furthermore, the components can be deployed on the Grid and then composed together dynamically at runtime. XCAT uses XML to describe the components and XSOAP, previously known as SOAP RMI[j], to communicate between them. Currently, XCAT is looking to implement Proteus (see Section 2.2.4) to achieve better performance and is working on using WSDL as a way to describe what kinds of communication mechanisms various components support.

2.2.3 File Systems and Storage

File systems and databases available today have the ability to satisfy a variety of the proposed system's requirements; such as, fault tolerance, versioning, persistence, and transparency. The systems evaluated here fall roughly into three categories; Distributed File Systems (DFS), Databases, and Federated File Systems. Among all the systems evaluated, Coda, FramerD, Objectivity, and Lustre offer the most potential.

Coda is an advanced distributed file system that has been designed around AFS, and therefore provides a global namespace. To be an effective distributed system, Coda has resolved the associated disconnectivity issue by providing client-side caching and client modification logs to update servers when a client reconnects. Though almost all OODB (Object Oriented Database) distributions offer these capabilities, what makes Coda unique is a sophisticated, optimistic, locking mechanism that provides conflict

[h] Electra, U Colorado, Xerox, JacORB, TAO, Jorba, omniORB, Mico, Arachne, ORBit (http://www.jetpen.com/~ben/corba/orbmatrix.html)

[i] An adaptation of Web Services technology has been realized in Grid Portals. Like the XCAT Science Portal and NASA IPG Launchpad, Portals provide an easily configurable GUI that can send control information to a running simulation. Complexity is masked from the user via the web GUI. Messages can be sent to a simulation with SOAP allowing the user to manipulate their application. Typically, these applications use a set of scripts to obtain a proxy certificate, configure the requested application for the user, and contact the application factory to generate an instance of the program.

[j] XSOAP is a C++, Java implementation of the SOAP protocol.

resolution and offers check pointing at the server level. Data is accessed at the file level and stored in Volumes on servers; object-level granularity is not supported. Other systems that were evaluated, but found less relevant, based on design purposes, were GFS, NFS, the Elephant File System, and Intermezzo.

As an alternative approach to storing massive amounts of data, Object-Oriented Databases come equipped with features like encryption and user authentication, versioning of classes, XML support, fault-tolerance through disconnectivity, client-side caching for performance, and automatic handling of updates. Common OODB's on the market are able to offer everything that Coda provides except for a similar conflict resolution scheme. Instead, OODB's typically approach conflicts with 'last guy wins.'

OODB's can approach operations in a distributed context by storing data in different volumes across multiple locations. Others, like Objectivity, approach data distribution in a federated environment, offering complete transparency across multiple servers and databases. In other words, to the user, the data appears and responds as if it were local. OODB's do not offer automatic data partitioning or redistribution.

Unlike a RDBMS (Relational Database Management System), Object-Oriented Databases are explicitly designed to handle objects and offer full support for concepts associated with the Object-Oriented paradigm such as encapsulation, inheritance, and polymorphism. The primary method for interacting with the database is through an Object-Oriented language, which can make accessing and operating on stored objects much easier for the programmer.

An OODBMS will provide object persistence and maintain integrity and concurrency of objects. In many OODB's, an application class can simply be declared "persistent capable". Issues like storing base and derived classes, linking to the schema, and working with referenced objects are all handled automatically. Some OODB's maintain their access permissions at the object level.

Although no OODBM found directly offers object-level versioning, OODBMS can support the concept. OODB's assign OID's to all objects in the database completely independent of the primary key. It would be possible to implement versioning by simply adding another field to the object schema and setting it to point to the known version OID. It may also be possible to extend the notion of a capability to reference the OID.

It was fairly difficult trying to locate a free OODB. Several projects and partial systems were found late in the survey and have not been fully explored. OZONE is an open initiative for the development of an open source Java-based Object-Oriented database system, still in the development stage. DB40 is an object database engine that has been implemented in Java and has been designed to work in a mobile environment. An even more promising system, which is currently available for download, is FramerD. Developed at MIT to support machine learning, FramerD is optimized for pointerintensive data structures that are often used by semantic networks, frame systems, and many intelligent agent applications. As a result, the design addresses what makes many applications obsolete in later years: the need to make an inflexible forward declaration of an object.

Federated systems are another storage possibility; these systems provide a global namespace and can pool together a vast number of storage resources into a common access source. The main benefit behind this approach is that it enables the system to scale both upward and downward to any degree. Possibilities for Federated systems are the Avaki Data Grid, the Lustre file system, and Objectivity's OODB.

Lustre is an open source, distributed, object-based file system that is being developed under ASCI. Lustre provides significant advantages over distributed file systems including: running on commodity hardware, using object based disks[k] for storage, and utilizing metadata servers for storing file system information. Replicated, failover metadata Servers (MDS's) maintain a transactional record of high-level file and file system changes. Distributed Object Storage Targets (OST's) are responsible for actual file system I/O and for interfacing with storage devices. Lustre supports strong file and metadata locking semantics to maintain coherency of the file systems even in the presence of concurrent access. File locking is distributed across the storage targets that constitute the file system. Each OST handles the locks for the objects that it stores. Lustre is due to be released at Lawrence Livermore on October 1, 2003.

Avaki is a commercial distribution of Legion. From a Federated perspective, Avaki is impressive in that it is able to carve up resources on one machine or millions. Individuals willing to contribute resources designate how much they are willing to donate to the pool, much like Legion. Avaki uses a capability-like system for access control and has been ported to a wide variety of architectures. However, the granularity does not extend to the level of an object.

The systems discussed here were discovered as being the optimum choices for a distributed system, a database system, and a federated system. Coda is not a likely candidate for use, simply because it has been surpassed by current implementations of OODB's and supports a limited variety of architectures. Lustre appears as though it will surpass Avaki as a Federated system with its regard for object-level granularity and parallel I/O. If an Object-Oriented database approach is to be taken, Objectivity appears to be the only to offer a federated view of the distributed resources. FramerD holds potential as a free OODB system that could be extended. Other systems evaluated in this

[k] Object-based Disks (OBD's) allow for better efficiency over conventional block access and storage.   An Object-based schema is applied to the disk that allows collections of data to vary in size as opposed to a predetermined or fixed block size.

category but not found applicable include GOOD's (Graph-Oriented Databases)[l]. Appendix D contains a comparison of the top three commercial OODB's.

2.2.4 Messaging and Communication

Many communication mechanisms were evaluated in terms of efficiency, interoperability, and their level of abstraction. It is apparent that no single protocol will be able to work effectively for HPC and at the same time be flexible enough to work with a highly heterogeneous collection of machines, such as the Internet. The need for a multi-protocol library seems essential in order to achieve both of these requirements. Object models such as CORBA and CCA offer one approach; however, they do not solve all interoperability and performance issues. These architectures can help provide persistence and maintain stateful communication, but they cannot be the only solution.

The trade-off in communications is fairly straight forward: to gain interoperability, performance must typically be sacrificed. Scientific floating point numbers are not well represented as text and can eat up extra bandwidth. However, it is possible to describe the communication protocols supported by two machines with XML and then let them determine which way is the most effective to transfer the data. Two machines that are interoperable through a high performance protocol like MPI should be allowed to communicate that way. Furthermore, if machine architectures are compatible then binary transfer should be supported.

A multi-protocol library should provide a common denominator protocol (e.g., SOAP), allow the user to switch between protocols dynamically, allow for the dynamic discovery of protocols, use metadata (XML for example) to define component interfaces, should allow for large, high–speed, data transfers, and should support automatic generation of stubs and skeletons. While it is clear that SOAP can meet many of these requirements, for high speed transfer it is not a good candidate. Java and C++ libraries like BinX, XDF, and XSIL offer support to the programmer for scientific data, but they are relatively new and have not been well tested. These libraries can offer a more canonical representation of scientific types that XML has a difficult time describing such as streams, tables and arrays [9].

PDBLib[m] (Portable Binary Database Library) and Proteus both address the interoperability issue associated with the transferring of binary data. Though binary data

[l] GOODB's (Graph-Oriented Database). Several Graph-Oriented approaches to databases are currently being pursued. These are in the early development stages and are mostly research projects. These systems use a front end schema to represent relational data in a graph structure. The data itself is not stored in this manner. Some projects are using OODB's as the backend for storage to achieve better efficiency.

[m] PDB Lib is an LLNL development that is part of PACT tools.

transfer is fast, it is not often applicable in heterogeneous environments. Since different architectures and machines represent binary data differently.

PDBLib follows the typical model for binary conversion (hub and spoke) where data is converted to a neutral format before passing, and the receiving end converts it back. PDBLib adopts this approach but only applies the conversion when necessary, to avoid the overhead of translating data when dealing with two systems that are binary compatible. PDBLib supports complex structures in two ways. It has a mechanism similar to the C struct and also handles pointers so that entire trees can be passed [10].

Proteus is a multi-protocol library for integrating different messaging protocols such as SOAP, JMS, and binary within one system. Clients developed with Proteus can communicate with any available messaging format without having to recompile. Proteus is written in C++ and is planning on porting to Java. Proteus objects act like a translator between two objects that speak different languages. A main design goal was performance and Proteus has addressed this issue by not requiring deserialization (packaging into objects) in Grids and by allowing for the intermediary object to pass data direct without buffering [11].

PBIO (Portable Binary I/O was also evaluated. PBIO offers higher performance but does so by eliminating DTD checking on the receiving end of the transmission. XML, CORBA, MPI, and RMI have been compared for a variety of performance and interoperability criteria in Appendix E.

2.2.5 Optimizations

Several applications have been discovered that can provide the more specialized, performance requirements of MObIUS, specifically, data partitioning and migration. It was difficult locating tools that operate with anything other than Fortran. Paradigm is one HPC compiler that stood out, but only operates on Fortran77 code.

Zoltan is a dynamic load-balancing library that has been developed and is in use at Sandia Labs. The Library has been written in C, uses MPI, runs on Unix, and has been designed to run on parallel computers and clusters of workstations. Zoltan offers automatic migration for objects that don’t have dependencies, like those in particle physics applications. In addition, Zoltan can be used as a parallel static partitioner for non-dynamic applications.

ParMETIS will automatically partition application data, both statically and dynamically, for parallel processing. ParMETIS provides a parallel library that partitions data based on a weighted graph. Nodes and vertices correspond to objects and have a weight that relates to their potential computational load. Edges on the graph correspond to communication costs.

Condor-G is a job scheduling application for HPC; it is an open source project developed at the University of Wisconsin. Condor-G has utilized the Globus Toolkit for security and communication; it runs on Linux, Solaris, Digital Unix, and IRIX. Condor-G is highly scalable, freely available, and is a currently evolving project. Although, it is regarded as one of the more sophisticated job schedulers (compared to Globus), it appears that very little of its functionality (load balancing, distribution, and migration) responds to the dynamic nature of the resources on which it operates.

2.3 Conclusion

The current state of distributed computing holds relevance to the development of MObIUS. There is a definite need for a general middleware that can address and adapt to a variety of architectures and maintain a high level of abstraction for the programmer. Grid, Peer, and Web Service technology are all merging toward a similar, general, infrastructure but have yet to achieve this goal. Based on the findings to date, there is no single system available that achieves all of the requirements we have suggested. Legion and Cactus-G are the best representatives of complete systems that could possibly be extended. Legion's object based design and native capability support make it a more suitable candidate for the proposed system. With adequate versioning capabilities implemented, Legion could be evolved into MObIUS.

Alternatively, if a new system is to be built, taking an existing application environment and extending it for a widely distributed environment with a toolkit like Globus, is one option. Third generation Peer applications like Tapestry, JXTA, and BOINC offer another alternative. With more efficient messaging algorithms, a natural ability to flex with the Internet, and user friendly API's, Peer applications offer great potential to the development of a general framework. Component architectures like CORBA or CCA can help provide the desired abstraction level to the programmer and interoperability among different platforms while natively supporting the notion of an object. Tools like Objectivity, FramerD, and Lustre can provide storage solutions for the proposed system. ParMETIS, Paradigm, and Zoltan can provide the more specific requirements such as data partitioning and data migration. Finally, Proteus, PDBLib, SOAP, BinX and similar XML libraries could be utilized in a multi-protocol communication implementation.

References

[1] I. Foster and A. Iamnitchi. *On death, taxes, and the convergence of peer-to-peer and grid computing*. In 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, CA, Feb. 2003.

[2] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. ReyCenvaz. *Programming the Grid: Distributed Software Components*, P2P and Grid Web Services for Scientific Applications. Cluster Computing, 5(3), 2002.

[3] Ian Foster, C. Kesselman, and S. Tuecke. *The anatomy of the grid: Enabling scalable virtual organizations*. International J. Supercomputer Applications, 15(3), 2001.

[4] http://www.cactuscode.org/Documentation

[5] *The Globus Project*. http://www.globus.org

[6] Thomas Sandholm, Jarek Gawr. *Globus Toolikt 3 Core – A Grid Service Container Framework*. July 2, 2003

[7] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz, *Tapestry: A Resilient Global-scale Overlay for Service Deployment*, IEEE Journal on Selected Areas in Communications

[8] Gary Kumfert. Lawrence Livermore National Laboratory, September 2003

[9] Martin Westhead, Mark Bull, *Representing Scientific Data on the Grid with BinX – Binary XML Description Language*. EPCC, University of Edinburgh

[10] *PACT overview*. http://www.pact.llnl.gov/Overview.html

[11] Kenneth Chiu, Madhusudhan Govindaraju, Dennis Gannon. *The Proteus Multiprotocol Message Library*. Indiana University, Computer Science Department, 2002

Appendix A, Complete Systems

S. G. Parker and C. R. Johnson, SCIRun: *A Scientific Programming Environment for Computational Steering*, in On-line Proceedings of the 1995.

http://www.cs.wis.edu/condor

G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. *Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus*. Proceedings of Supercomputing 2001.

http://www.cactuscode.org/Documentation

A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds. *Legion: The Next Logical Step Toward a Nationwide Virtual Computer* CS 94 -21, University of Virginia, 1994.

A. S. Grimshaw, A. Nguyen-Tuong, And W. A. Wulf, *Campus-wide computing: Early results using legion at the University of Virginia*, Tech. Report CS-95-19, University of Virginia, Mar. 1995.

http://legion.virginia.edu

Madhusudhan Govindaraju, Sriram Krishnan, Kenneth Chiu, Aleksander Slominski, Dennis Gannon, *XCAT 2.0: A Component-Based Programming Model for Grid Web Services*. Indian University, Bloominton, IN, Department of Computer Science –

www.extreme.indiana.edu/xcat/publications/tr-xcat.pdf

James Frey, Todd Tannenbaum, Ian Foster, Miron
Livny, and Steven Tuecke, *Condor-G: A Computation Management Agent for Multi – Institutional Grids*.

www.cs.wisc.edu/condor/condorg/

Avaki Poised to Take Legion and Commercial Grid Computing to the Edge.
2001 Online: News about the NPACI and SDSC Community, Volume 5, Issue 12 - June 13, 2001

Appendix B, Peer-to-Peer Systems

D. Milojicic, V Kalogeraki, R. Lukose, K Nagaraja, J Pruyne, B. Richard, S Rollins, Z. Xu, *Peer-to-Peer Computing*, Technical Report HPL-2002-57, HP Labs. 2002 http://www.objectstore.net

http://www.jxta.org

http://www.gotdotnet.com

http://www.oceanstore.cs.berkeley.edu

http://www.boinc.berkeley.edu

http://www.research.microsoft.com/~antr/pastry/

Ian Clarke, *A Distributed Decentralized Information Storage and Retrieval System*, University of Edinburgh, Division of Informatics, 1999

D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. *Resilient overlay networks*. In Proceedings of SOSP '01, Banff, Canada, Oct. 2001.

Appendix C, File Systems

P. Braam, M. Callahan, and P. Schwan. *The Intermezzo File System*. In Proceedings of the 3rd of the Perl Conference, O'Reilly Open Source Convention, Monterey, CA, USA, Aug. 1999.

Bill von Hagen. *Distributed Filesystems for Linux*. Linux Magazine, November 2000

D. S. Santry et. al. *Deciding when to forget in the elephant file system*. In Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles, pages 110--123, December 12-15, 1999.

M. Satyanarayanan, *The Evolution of Coda*, ACM Trans. Computer Systems, vol. 20, no. 2, May 2002, pp. 85–124.

Peter J. Braham. *The Coda Distributed File Syst*em. Linux Journal, June 1998.

Appendix E, Communications

C. Reichenberger, *VOODOOA Tool for Orthogonal Version Management*, Software Configuration Management: Selected Papers SCM-4 and SCM-5, pp. 61-79, Apr. 1995.

Java.sun.com *Introduction to CORBA Short Course*. –
http://www.developer.javea.sun.com/developer/onlineTraining/corba/corba.html#co5

S. Vinoski. *New Features for CORBA* 3.0. Communications of the ACM, vol. 41, pp. 44--52, October 1998.

The DOE Common Component Architecture Project –
http://www.extreme.indiana.edu/~gannon/cca_report.html

Fei Sophie Gao.*The Comparison of CORBA and SOAP*. ITC UAH, February 28, 2002

eXtensible Data Format (XDF) Homepage,
http://www.xml.gsfc.nasa.gov/XDF/XDF_home.html

M. Govindaraju et al. *Requirements for and evaluation of RMI protocols for scientific computing*. In Supercomputing, 2000.

http://www.research-indiana.org/iu_proteus.html

Bierman, G. M. *Using XML as an Object Interchange Format*, Department of Computer Science, University of Warwick, May 17, 2000.

Why use MPI?. http://www.mpi-softech.com, June 2003

What is MPICH-G2? http://www.hpclab.niu.edu/mpi/g2_body.html

F. E. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener, *Efficient wire formats for high performance computing*, in Proceedings of Super- computing '00

Dan Wahlin, *Top 5 Uses for XML*. XML & Web Services Archives.
http://www.fawcette.com/xmlmag/2002_01/online/online_eprods/xml_dwahlin01_18/default.asp

Works Consulted

Storage

Ullman, J.D., *A comparison Between Deductive and Object-Oriented Database Systems*, Proc. of the Int'l Conf. on Deductive and Object-Oriented Databases, 1991, pp. 263-277

Steve McClure. *Object Database vs. Object-Relational Databases*., IDC Bulletin #14821E, August 1997

Ronald Bourret. *XML and Databases*. http://www.rpbourret.com/xml/XMLAndDatabases.htm#xmlquery, July 2003

http://www.fastobjects.com

Capabilities

Mullender, S. J. and Tanenbaum, A. S. (1986). *The design of a capability-based distributed operating system*. The Computer Journal, 29:289-299.

T. Riechmann and F. J. Hauck, *Meta Objects for Access Control: Extending Capability-based Security*, In Proceedings of New Security Paradigms Workshop, Langdale, Cumbria, UK, 1997, pp. 17-22.

R.S. Fabry, *Capability-Based Addressing*, CACM, vol. 17, no. 7, pp. 403-412, July 1974.

Grids and Distributed Architectures

J. Nick I. Foster, C. Kesselman and S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

W. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of*

Large Scientific Datasets, Journal of Network and Computer Applications, 23:187-200, 2001.

Geoffrey Fox Grids and Peer-to-Peer Networks for e-Science
at SPECTS 2002 meeting San Diego CA July 17 2002;
http://grids.ucs.indiana.edu/ptliupages/publications/presentations/

http://www.teragrid.org/

Roxana Diaconescu, Reidar Conradi, *A Data Parallel Programming Model Based on Distributed Objects*. Norwegian University of Science and Technology, Computer and Information Science Department

O'Reilly Network, http://www.oreillynet.com

Complete Systems											
system	**platforms**	**language support / interoperability**	**supports legacy code**	**transparency**	**scalability**	**access control / capabilities**	**object / data distribution**	**object migration (failure)**	**object migration (forced for optimization)**	**data partitioning (load balancing)**	**data partitioning (parallelization)**
Legion	Windows, Linux, Solaris	Excellent	YES (Fortran, Ada, C, and others)	High	Excellent	YES	YES	YES	No Data	No Data	YES
Cactus-G	Linux, Windows 2000 &NT, Alpha Linux, Alpha OSF, SGI, IBM SP, MacOS X, Fujitsu, Hitachi SR8000-F1, Sun Sparc, Cray T3E, Mac Linux, Open BSD	F77, F90, C, C++, also cactus development language for developing parallel applications, Requires Globus	YES	High	High	N /A	YES	No Data	No Data	No Data	YES
Amoeba	Sun-3, 4, Micro Sparc, 3/60 &3/50 workstations, Intel Pentium, mostly POSIX compatible, some UNIX interoperability	ANSI C, Pascal, Modula 2, GNU BASIC, Fortran 77, Orca (parallel programming lang)	Limited languages	High	No Data (estimated to be Good)	YES	YES	YES	YES	No	No data
NLTSS	Legacy supercomputers at LLNL: Cray, CDC 7600	an alternative to UNIX, supports LTSS	YES	YES	High	YES	YES	NO, in event of failure, memory dump to disk	NO	NO	No data

The information used here has been collected from a variety of documented sources and interviews. For a list of sources that contributed to this sheet please refer to *References, Appendix A, Complete Systems* .

Complete Systems											
system	**persistence**	**replication**	**updates replicants**	**versioning of objects**	**state of system**	**developer**	**cost / licensing**	**extendibility**	**design**	**job scheduling**	**performance**
Legion	YES	YES	YES	NO but does provide check pointing and supports concept of object states	up and running, continuing progress	University of Virginia	Unsure	federated file system was designed to permit extension of basic services	Object Oriented layered virtual machine	YES	High
Cactus-G	YES, check pointing	NO	N /A	CVS for development of new thorns	implemented and on going, new thorns are being developed	Open Source research project	Freely available	YES, modular design allows for new functionality to be developed as a thorn, implemented in C++, C and perl	modular framework for Grid Environment	YES	High
Condor-G	YES, Check pointing of job queue	NO	N / A	N /A	implemented and ongoing	university of Wisconsin	Freely available	No Data	No Data	Excellent	High
Amoeba	YES	YES	YES	NO	most recent data from 1996	Vrije Universiteit, (Amsterdam)	Freely available for research institutions and universities	the OO design of the system allows for some interchangeability I.e. File servers can be swapped	distributed, parallel, operating system	YES	Designed for high performance (FLIP protocol)
NLTSS	YES	YES, data moved directly to processor for high performance	No Data	No data	retired from use at the lab in 1993, replaced by UNIX	LLNL	FREE to government agencies, but may not be complete	No Data	Pure message-passing, Network Operating System	YES	Extreme

The information used here has been collected from a variety of documented sources and interviews. For a list of sources that contributed to this sheet please refer to *References, Appendix A, Complete Systems* .

Appendix A, part 2 – Complete Systems

Peer-to-Peer Systems								
system	**technology**	**communication / messaging / lookup**	**anonymity(P, R, S, D)**	**security**	**decentralization**	**supported applications**	**scalability**	**availability / developer**
.NET	platform	SOAP, XML, COM	N/A	passport, cryptography, author/auth, support for SSL, Kerberos, others	client/server and pure P2P	Many, including MS Office	world-scale	Microsoft
JXTA	platform	XML / JXTA search	N/A	cryptography algorithm, distributed trust model	client/server and pure P2P	limited	addresses embedded systems	Sun Microsystems public domain
Tapestry (Java)	overlay	KBR (Key-Based Routing based on DHT), DOLR	N /A	PKI trust, message authentication support	distributed, no central manager	OceanStore(global persistent data Store that pro-actively migrates data), Bayeux(multicast distribution)	Designed for billions	UC Berkeley
Pastry(C#)	overlay	KBR (based on DHT), DOLR	N /A	certificates for nodeID, cryptography algorithm	distributed, no central manager	SCRIBE: group communication, PAST archival storage, SQUIRREL: cooperative web-caching, SplitStream: content distribution, POST: co-operative messaging	Designed for billions	Microsoft Research
Gnutella	file sharing	(http) broadcast with TTL	Multicasting, covert paths (P)	not addressed, no encryption support	hybrid	content distribution, Lime Wire	thousands, has the potential to flood (broadcast messaging)	open source
FreeNet (Java)	file sharing	KBR (based on DHT)	covert paths, identity spoofing(P)covert paths (R)non-voluntary placement(S), encryption(D)	full anonymity & prevention of DoS	pure P2P	content distribution	log(network_size)	open source
Napster	file sharing	contact central server, download form peer	N/A	N / A	centralized	content distribution	millions	not sure
Groove	collaboration / platform	XML	poor	shared-spaces, authentication/authorizatio n, encryption	hybrid	purchasing, inventory, auctions, messaging	N/A	not sure
Magi	collaboration	XML	N/A	certificate authority	hybrid	instant messaging, shared file access, chat	100 corporate networks ?	not sure
Avaki	distributed computing / overlay	IDL, variety of bindings, supports mpi	N/A	encryption, authentication	distributed, no central manager	computation grid, shared(federated) secure data access	scale to 1000's (2.5 - 3k tested)	Avaki
SETI	distributed computing	download from server	medium	N / A	master/slave	closed	millions (highly scalable server side)	UC Berkeley

* Anonymity refers to P= publisher, R = reader, S= Server, D = document; DOLR = Decentralized Object Location and Routing messages are mapped to virtual 'endpoints' which enables message delivery in the presence of network failure.

While there are hundreds of peer systems today, the ones chosen here represent prominent examples from each peer category. Categories of peers include platform, overlay, file sharing/collaboration, and distributed computing. The information used here has been collected from a variety of documented sources. For a list of sources that contributed to this table please refer to *References, Appendix B, Peer-to-Peer Systems* .

Appendix B, part 1 – Peer-to-Peer Systems

Peer-to-Peer Systems

system	platform	distinctive features	fault tolerance	guaranteed delivery	persistence	replication	performance / optimization (caching)	load balancing (object migration)
.NET	Windows	widespread MS application base, multi-language interoperability, extensive API's, designed for web applications	High	YES	YES	YES	No Data	YES
JXTA	Windows, Linux, Solaris	open source, platform independent	High	YES	YES	YES	No Data	YES
Tapestry (Java)	JVM	structured and symmetric, very resilient to dynamic changes	Yes, adaptive algorithms	Yes in an operating network and Yes, even through fail-over (DOLR)*	application specific objects	supports application specific object replication, application specific object caching	YES hot spot caching (publishes pointers)	YES applications are allowed to place objects where it needs them
Pastry(C#)	Windows	structured and symmetric, very resilient to dynamic changes	Yes, supported	Yes in an operating network and Yes, even through fail-over (DOLR)*	notifies applications of changes in dynamic environment	supports application specific object replication, application specific object caching	YES	Yes, application Independent
Gnutella	Windows, Linux	free protocol(Gnutella is a model), large user community	resume download	NO	YES, resume download	YES	No Data	No Data
FreeNet (Java)	any JVM system	preservation of anonymity, written in Java	decentralization, replication	NO	NO	Some	High / routing table built on cache	No Data
Napster	common consumer OS supported	centralized server	server replication	NO	NO	server replication	Lookup server w/ P2p download	N / A
Groove	Windows	self updating, multiple identities, follows COM model	queued messages	YES	No Data	No Data	No Data	N / A
Magi	Windows, Mac	handhelds	queued messages	YES	No Data	No Data	High	N / A
Avaki	Windows, Linux, Solaris	based on Legion, heterogeneity, secure access, high parallel execution, distributed administrative control, highly transparent(masks complexity)	check pointing, reliable messaging, m migrates failed jobs	YES	YES	caching	High, even on platforms with different communication characteristics	Yes
SETI	All common user OS supported	resilience = check pointing on users disk, very large scale	timed check pointing, redundancy, reissues jobs	through central server	Yes, writes to local hard drive	Yes, replicates data sets and jobs, for accuracy	High (based on parallelism)	No Data

* Anonymity refers to P= publisher, R = reader, D = document, S= Server, DOLR = Decentralized Object Location and Routing messages are mapped to virtual 'endpoints' which enables message delivery in the presence of network failure.

The systems described in this chart have been evaluated on a variety of criteria that relates to the requirements of MObIUS. While there are hundreds of peer systems today, the ones chosen here represent prominent examples from each peer category. Categories of peers include platform, overlay, file sharing/collaboration, and distributed computing. The information used here has been collected from a variety of documented sources. For a list of sources that contributed to this sheet please refer to *References, Appendix B, Peer-to-Peer Systems* .

Appendix B, part 2 – Peer-to-Peer Systems

File Systems

system	platform	scalability	transparency	client side caching	handles disconnectivity	security	licensing	access to storage	notes
NFS	Unix, Linux, Windows	Fair	No Global namespace	Yes	No	Poor	Linux = GPL, others proprietary	Syscall-level access	very popular, primitive locking mechanisms, no client side cache makes it stateless
InterMezzo	Linux Kernels 2.2 and up	Good	global namespace	Yes	Yes, persistent cache	No	GPL	File	inspired by, but not based on CODA source, uses a journaled file system (stores changes in a log), can interoperate with other journaled file systems
AFS	Unix, Red Hat Linux, Windows	Excellent	global namespace	Yes	No	Kerberos	IBM Public License	File	pioneered persistent client-side caching, doesn't offer persistence through disconnectivity
CODA	Linux, NetBSD, FreeBSD, Windows	Very Good	global namespace	Yes	Yes, persistent cache	Kerberos, access control lists	GPL, LPGL (Lesser GPL)	File, Volume	built upon AFS but handles disconnectivity
GFS	Linux Kernels 2.2 and up	Good	No Data	No	N/A	Any	GPL	Block	for Linux clusters, accesses storage at the block level, makes this system inherently processor and system independent, rather than file; open source SAN

This table provides a rough comparison of the NFS, InterMezzo, CODA, GFS, and AFS file systems. The data used in this chart has been collected from several different documents. For a list of references from which the data was derived, please refer to *References, Appendix C, File Systems* .

Object-Oriented Databases

PRODUCT CRITERION	FASTOBJECTS t7 9.5	VERSANT (6.0)	OBJECTIVITY
versioning	schema level versioning (on-the fly), class level	(on the fly) schema level versioning, updates	object, class, schema
replication	Hardware level (clustering), server level replication (asynch and synch)	server level replication	synchronous server level across geo. dispersed servers
security / encryption	data block encryption, SSL client-server (blowfish is default but also supports plug-ins)	No Data	no encryption, SSL client server (Kerberos is default but also supports plug-ins)
language support	Native C++ and Java, C++ (transparent access)	Native Java and C++, C++ (transparent access)	C++, Java, Smalltalk, SQL 92 interface
XML support	import and export, have DTD defined	import and export toolkit	NO (import /export expected in upcoming release 8.0)
system platforms	Windows(all), Solaris, HP Unix, VX Works, AIX, Red Hat Linux	NT, 2000, Solaris, AIX, HP Unix, Red Hat Linux	Windows, Unix, Linux Digital HP, IBM, Intel, Silicon Graphics, Sun
availability	fail-over, replication	fail-over, replication	fail-over, replication
load balancing	achieved through architecture (master / slave + caching) (achieved through replication)	achieved through replication	automatic when new clients are added, and through replication
scalability (memory / architectural)	2GB	64 bit support	Federated Database (exabytes), one installation can support 1000 clients per server, 64 bit support
OODBM Interoperability	ODBC (objects put in tables) OQL, JDOQL	ODBC	No but much of ODMG is supported
RDBMS Interoperability	SQL Object Factory (POOR Performance)	VQL(Versant Query Language), SQL, JDBC	SQL, JavaBeans
integration with CORBA	NO direct support but can be used in conjunction with CORBA	No Data	NO direct support but can be used in conjunction with CORBA
caching	writes to server will be propagated to client on a read (maintains coherency)	dual server caching	client-side based on pages (LRU)
data transfer method	Object	No data	Pages
object level locking	Yes, pessimistic	Yes	Yes, pessimistic
notes	spoke with an engineer when gathering data	tools: XML toolkit, object Inspector, Monitoring console, Administration console	spoke with an engineer when gathering data
persistence	achieved at creation time	No data	achieved at creation time
fault tolerance	Yes	No data	Yes, check pointing with online incremental back-up utility

This table provides a detailed comparison of three current versions of OODB's that are on the market. The data used in this chart has been collected from product engineers and related whitepapers.

Communication Protocols

communication protocol	uses	ease of use / abstraction level	interoperability (disparate systems)	security	object access via references	performance	distributed garbage collection	transaction state maintained (persistence)	data transmission	drawbacks	advantages
CORBA	distributed service processing and object communication across a variety of environments	Poor	Moderate	questionable	YES	Good (avg. 3x's faster than soap)	Supported	YES	binary encoding	security is an issue no support for versioning, just a standard	Highly transparent
RMI	Java API	Good (Java API)	Java only	Good	YES	Very Good	YES	YES	binary encoding	need java compiler to take advantage of type checking benefit	can offer type checking, exception handling, part of Java API
MPI	High Performance Data Transfer	Fair	Poor	N/A	NO	Excellent	NO	N/A	binary encoding	extremely rigid	fast
SOAP	Messaging	Excellent	Excellent	questionable	NO	Poor	NO	NO	Unicode	lacks transaction management, server needs to timeout the object to reclaim memory, requires a greater amount of run-time checking	great for asynchronous communication and loosening the degree of coupling between client and server

The table provided here shows the relationship performance and interoperability among data transfer methods. As interoperability increases, performance typically declines. References to the sources from which this data was obtained can be found in *References, Appendix E, Communication Protocols* .