Architecture and Implementation of Distributed Data Storage using Web Services, CORBA and PVM[⋆]

Pawel Czarnul

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology, Poland,
pczarnul@eti.pg.gda.pl, http://fox.eti.pg.gda.pl/∼pczarnul

Abstract. We propose an architecture and its implementation called PVMWebCluster I/O targeted for distributed data storage and retrieval. Data is submitted via Web services from geographically distant clients or via CORBA from within clusters which offers better performance. The system consists of many, possibly geographically distributed clusters which run DAMPVM/PVM. Each cluster has a manager waiting for CORBA read/write calls which are passed to particular nodes in the cluster via PVM messages. Each cluster has a corresponding Web service that passes read/write calls to the CORBA manager. Web services form a top layer of PVMWebCluster I/O and call each other to obtain the best cluster/node to store a particular chunk of data. This results in a very flexible architecture which integrates distributed clusters for data storage of small and large files. The architecture proposes caching at both Web service and cluster layers. We have tested an initial implementation by submission of files of varying sizes in four configurations: via the Web service and CORBA layers on a local machine, via Web services in a LAN and via Web services through the Internet.

1 Introduction

In recent years, the need for parallel and distributed computing has increased significantly. This refers to both high performance computing within tightly coupled clusters using MPI and PVM ([1]), grid architectures ([2]) and Internet based multi-tier technologies ([3]) like J2EE, servlets, JSP etc. As high performance computing processes huge amounts of data, efficient means of handling it is necessary. We present the evolution from high performance parallel file server based architectures to highly distributed Web based data storage in XML. The proposed PVMWebCluster I/O uses Web services ([4]) and CORBA ([5]) as interfaces and PVM ([1]) within clusters.

2 Related Work

There are many systems in the literature targeted for parallel and distributed data storage/retrieval. However, they seem to be solutions either too focused on tightly coupled parallel computations like implementations of MPI-I/O ([6]) or are very general like:

⋆ work partially sponsored by the Polish National Grant KBN No. 4 T11C 005 25

WWW-based systems like Metacat ([7]) or OceanStore ([8]) or grid systems like EU-DataGrid ([9]) based on Globus ([10]) or GridLab ([11]). The latter ones are general solutions which integrate remote job control, data management and several other services. The proposed PVMWebCluster I/O is a dedicated solution for a collection of distributed PVM-based clusters so specific solutions like submission from within a cluster can be optimized considerably using the loads by other user processes, dynamic requirements reported by DAMPVM processes etc.

Network file systems like NFS or AFS ([12]) are easy to use within clusters as they support one uniform file tree for all the applications. However, integration of such systems in various clusters is not easy and requires additional tools.

MPI-I/O ([6]) is a set of parallel access operations to files defined within MPI-2 and thus limited to MPI. There are systems which support interoperability and file controlled sharing for applications running on different Massively Parallel Processors (MPPs). This enables different MPI vendor implementations to interoperate like in MPI_Connect ([13]). MPI_Conn_IO API allows file access from parallel applications running on different parallel computers. MPI_Conn_IO API is used to open a file globally and split it across the parallel sites, then possibly open it using the MPI-2 MPI_File_open() and update it. [14] proposes Stampi-I/O – a distributed parallel I/O library, an implementation of MPI-I/O that supports parallel file read/write operations. Another similar system is PACX-MPI PIO ([13]) in which many clients can access many parallel file servers. PIOUS ([15]) is a similar system for PVM which implements a virtual file system within a PVM environment and thus is a parallel not a distributed system by our definition.

The following systems support distributed file storage but their architectures do not seem to offer any specific optimization possibilities for PVM clusters with changeable loads and shared by many users at the same time as in the case of PVMWebCluster I/O. [16] presents the architecture of WebFS – a cache coherent distributed file system for unmodified applications which uses global HTTP naming to write and retrieve files. UFO ([17]) is another user level implementation of a distributed file system in which remote files can be treated as if they were local and are accessed using FTP or HTTP.

The following two systems are again too general and high-level in their architectures to be used efficiently in HPC applications on a collection of PVM-based clusters. As an example, OceanStore ([8]) has been developed with thousands of users and terabytes or more of data in mind. It is highly dispersed across the Internet and consists of distributed pools of storage systems each of which consists of particular servers. [7] presents Metacat – a framework for distributed data storage that is physically distributed across the Internet, possibly heterogeneous with respect to data format. Data is stored in XML in SQL-compliant relational databases.

Regarding the latest developments, there are grid-based systems like EU-DataGrid ([9]) and GridLab ([11]). However, since the specifications focus rather on the requirements, APIs, portability and ease of integration of various systems, there is no specific mention of PVM/MPI optimizations as the systems focus on high-level distributed job control of large applications instead. The Global Access to Secondary Storage system (GASS, [18]), a part of the Globus toolkit ([10]) used in grid-based systems, is a data access and movement service that uses URLs to implement a global file space. GASS provides cache techniques for read and write operations.

3 PVMWebCluster I/O Architecture

PVMWebCluster I/O is based on the three-tier PVMWebCluster architecture which corresponds to the following three layers in PVMWebCluster I/O:

1. Web Service Data Submission Layer (WSDSL) – a geographically distributed PVMWebCluster system is composed of particular clusters each of which has a corresponding Web Service interface. Web Services representing separate clusters call each other to determine the best cluster to store a particular chunk of data.
2. Cluster Data Submission Layer (CDSL) – each cluster has a CORBA representative that intermediates incoming data submission calls to the cluster as well as returns information such as: available storage space, processor speeds, internode latency and bandwidth, number of active processes on particular nodes etc. This information can be used by the WSDSL to make a decision in which cluster to store a particular chunk of data. The cluster manager listens to CORBA calls and uses PVM communication to store the data on one of the available nodes in the cluster. The CDSL has a database of files submitted to it. Such submission is faster but data can be partitioned and spread using only the cluster nodes. On the other hand, using the WSDSL, data can be replicated and stored on physically distant clusters in case one cluster is damaged.
3. Cluster Layer (CL) – finally a chunk of data is stored on a certain node in the cluster. The data is submitted from the CDSL, currently implemented as PVM messages.

The proposed system architecture of PVMWebCluster with distributed file storage PVMWebCluster I/O is presented in Figure 1. We distinguish the following data submission modes for both distributed and sequential processing:

1. Large data submission through Web Services – used to store large amounts of data which does not require frequent and fast access:
   - this solution involves large latency both when data is submitted and retrieved,
   - allows really large data capacity as the system architecture includes a set of clusters each of which consists of many nodes equipped with disks.
2. Storage for high performance computing within clusters – reasonably small amounts of data which is to be accessed reasonably frequently. In the initial implementation, we assumed that in this submission mode files would not need partitioning. However, it may be useful to replicate files across the nodes in the cluster so that processes can access them in parallel.

4 Data Access Patterns, Submission and Partitioning in PVMWebCluster I/O

WebFS implements three cache coherence policies:

1. Last writer wins – a server keeps a listing of all sites caching the given file. If it has been updated by one of them, invalidation notifications are sent to the others. This makes this policy reasonable for occasional updates rather than frequent updates of many files.

2. Append only – writes append information to files and can be simply forwarded to other servers and receivers.
3. Multicast – all updates are sent to all clients. One dedicated channel is used to distribute invalidation (update) notifications while another one for sending updates.

In GASS ([18]), common access patterns have been distinguished and implemented to achieve low latency access to files as well as high bandwidth. They include: read-only, last writer wins, append-only and unrestricted access to remove the need for parallel call synchronization.

PVMWebCluster I/O is more about distributed file storage i.e. file partitioning and distribution rather than multi-user access. In PVMWebCluster I/O, we assume that a file is generally used by the user who has written it. However, extensions are possible in the future thanks to the flexible architecture. As in OceanStore ([8]), we assume that any update of a file is a new version of it and is stored as a new file. It essentially eliminates the need for concurrent write control for files. The URL global space is naturally used as the implementation is based on Web services. There is a dedicated directory for data storage on each node.

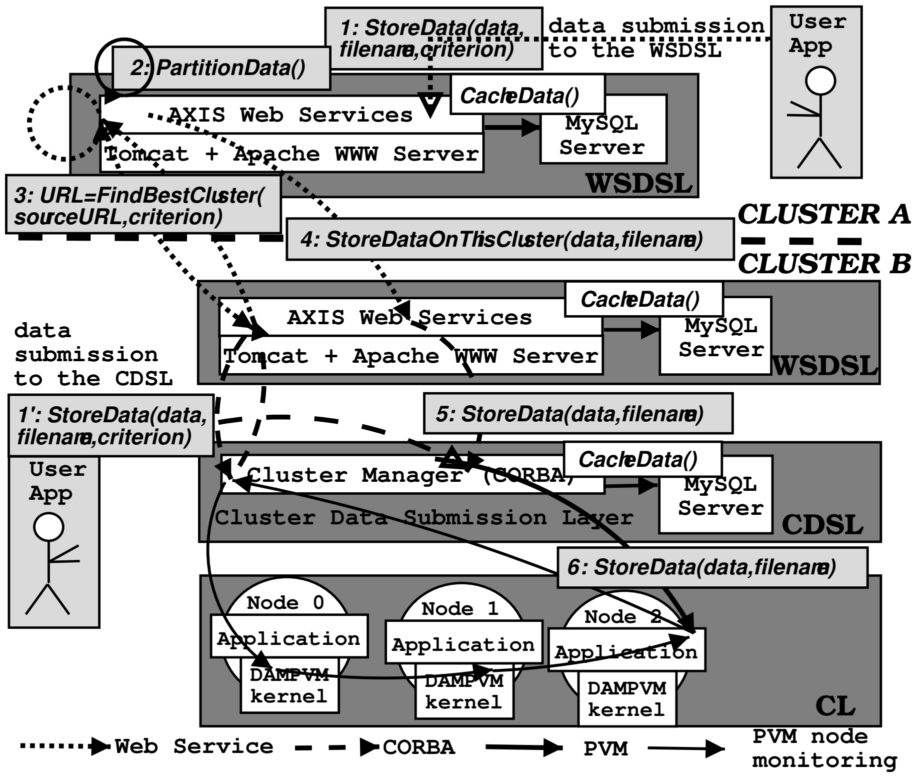
1: StoreData(data, filename,criterion)
data submission to the WSDSL
User App
2: PartitionData()
CacheData()
AXIS Web Services
Tomcat + Apache WWW Server
MySQL Server
WSDSL
3: URL=FindBestCluster(sourceURL,criterion)
4: StoreDataOnThisCluster(data,filename)
CLUSTER A
CLUSTER B
data submission to the CDSL
1': StoreData(data, filename,criterion)
5: StoreData(data,filename)
Cluster Manager (CORBA)
Cluster Data Submission Layer
CDSL
6: StoreData(data,filename)
Node 0
Node 1
Node 2
Application
DAMPVM kernel
CL
Web Service
CORBA
PVM
PVM node monitoring

Fig. 1: PVMWebCluster I/O Architecture

The write submission requests to both the WSDSL and the CDSL are shown in Figure 1. For the WSDSL the steps are as follows:

1. A write call is invoked in the client code. Assuming the storage of data in a string, the client code invokes Web service `public static String StoreData (String sData, String sFileName, int nCriterion)`. The data is distributed among the available clusters. The information about all the blocks, their numbers and locations is stored in a MySQL database. In the experiments, single data submissions were not partitioned across nodes. In this case, the Web service returns the URL of the final Web service, the node within the cluster where the data has been saved and the available free space (in KBs) on this node (`<BestService URL>>><BestNodeName>>><result>` e.g. `wolf.eti.pg.gda.pl>>wolf>> 10773.836`). The criterion given as an argument determines the algorithm used. The advantage of the Web Service technology is the simplicity of the client code.
2. The Web service decides whether the data size is too large and thus should be partitioned into chunks. If this is the case, the data is partitioned by method `Partition Data()`. Then the following operations are executed on the chunks in parallel.
3. For a data chunk, invoke method `public static String FindBestCluster (String sSourceServiceURL, int nCriterion)` which finds the best (with respect to the criterion set, in this case `FILE_STORAGE_AVAILABLE_MAXIMIZATION_CRITERION` since we are looking for maximum available disk space) cluster in the subgraph of the system. The clusters being called memorize the URL and the parameters of the best Web service/node found so far. The source URL is given in order not to call it back recursively. The cluster which returns the maximum available storage space is assigned the data. On every cluster method `FindBestCluster()` invokes method `GetClusterMaxFreeFileStorageSpace()` which returns the file storage (corresponding to the node with the maximum free space in the cluster). This method uses the available storage space for every node within the cluster that is cached at the WSDSL layer. It is gathered by another thread working concurrently which makes CORBA calls to the CDSL layer cluster manager in parallel.
4. Invoke Web service `public static String StoreDataOnThisCluster (String sData, String sFileName)` which passes the data to the optimal cluster selected in the previous step.
5. On the final cluster, a call is made to the `StoreData()` CORBA method in the CDSL layer cluster manager. Its Interoperable Object Reference (IOR) is fetched from a file written to the disk during the initialization of the server. The file name and the data are passed. This layer can also be called by an application from within the cluster via CORBA (1'. in Figure 1).
6. Finally the CORBA manager stores the data on the best node in the cluster i.e. the node with maximum available free storage space for the aforementioned criterion and sends the data via PVM messages.

Figure 1 shows the cache procedures (currently being implemented) at the WSDSL and CDSL levels. Subsequent read operations can use the cached data. The cache at the CDSL level can contain more information than the WSDSL cache as some files may have been submitted to the cluster manager via CORBA calls from within the cluster.

5 Implementation and Experimental Results

Entry points to clusters have been implemented as Web services in Java with the AXIS server (published as `.jws` files, [4]) running in the Tomcat application server ([19]). AXIS is a SOAP engine and Tomcat runs on the Apache WWW server. The architecture allows PVM clusters running on different user accounts.

We have implemented read and write operations through Web services (implemented in Java) and then via CORBA calls to particular clusters. Cluster managers (implemented in C++) monitor performance parameters including available storage space from all the nodes in the clusters they are in charge of. This is done by DAMPVM kernels ([20], [21], [22]), previously developed by the author. They use PVM communication and OS system calls. The following four configurations were tested:

CORBA – local machine – data is written to a local machine via a CORBA call. A Pentium 4-M 1.4GHz workstation running Redhat Linux 8.0, kernel 2.4.18-18.

Web service – local machine – data is written to a local machine through a Web service (the WSDSL level). The WSDSL layer contacts the cluster manager at the CDSL level which writes the data in the cluster. Same configuration as above.

Web service – through LAN – data is written to a machine in a local network through a Web service (the WSDSL level) which invokes Web services on other nodes to determine the cluster with the largest available storage space. Then a Web service is invoked on the chosen cluster which contacts the cluster manager at the CDSL level which writes the data in the cluster. Pentium 4-M 1.4GHz and Athlon XPs 1800+ workstations running Redhat Linux 8.0, kernel 2.4.18-18, 10Mbps Ethernet.

Web service – through Internet – data is written to a distant machine through the Internet via a Web service (the WSDSL level). The communication is done as in the previous configuration with the exception of much larger latency and lower bandwidth through the Internet. Same configuration as above through the Internet (one node connected to the Internet via a shared 115kbps SDI connection).

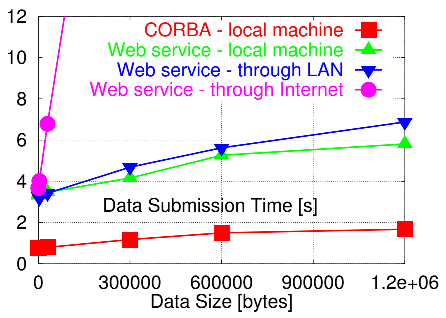

Fig. 2: Write Times with Client Overhead

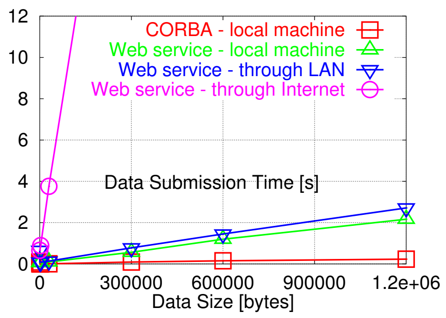

Fig. 3: Write Times without Client Overhead

It is must be noted that data was submitted through Java clients that read it and passed to the WSDSL or CDSL layers respectively. Figures 2 and 3 show the obtained results for submission of files of the following sizes: 30, 300, 3000, 300000, 600000 and 1200000 bytes. Figure 2 shows the times of single write calls via the client in which case we must account for the client initialization as well. This corresponds to occasional write calls when the initialization overhead is significant. The results are averaged from 10 write calls. Figure 3 shows the write times of successive write calls after the client has already been initialized and the first call is already finished. The results are averaged from 100 subsequent calls without client initialization. This corresponds to many subsequent submissions for which the initialization overhead can be neglected.

Figure 2 shows that the overhead for invoking a client and thus the initialization of necessary Web service and CORBA components is considerable, even for small 30-byte files. We also conclude that this initial overhead is much smaller when data is submitted within the cluster through the CORBA manager than through Web services. Additionally, Figure 3 shows that the Web service implementation puts a limit on the bandwidth even on a local machine compared to the CORBA communication. However, it is also shown that in practice this may not be a determining factor as the low Internet bandwidth is the bottleneck for distributed data submission.

6 Summary and Future Work

We have proposed and implemented a system for distributed data storage and retrieval based on Web services as the top layer, CORBA as middleware and PVM inside clusters. The performance of the implementation for files of varying sizes have been assessed for submissions via both the Web service and CORBA layers. Although the system is fully functional and complements PVMWebCluster in distributed task execution and management, there are many features to be implemented in PVMWebCluster I/O:

– various partitioning and caching techniques for very large files to be stored, tuning parameters and performance measurement for large networks,
– data encryption and decryption for distributed data storage in open environments,
– integration with the PVMWebCluster user management,
– data replication and migration to increase bandwidth,
– node/cluster/network failure handling.

References

1. Wilkinson, B., Allen, M.: Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Prentice Hall (1999)
2. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications **15** (2001) 200–222 `http://www.globus.org/research/papers/anatomy.pdf`.
3. Noack, J., Mehmaneche, H., Mehmaneche, H., Zendler, A.: Architectural Patterns for Web Applications. In Hamza, M., ed.: 18th IASTED International Conference on Applied Informatics (AI 2000), Proceedings, Innsbruck, Austria, ACTA Press (2000) `citeseer.nj.nec.com/260788.html`.
4. Streicher, M.: Creating Web Services with AXIS: Apache's Latest SOAP Implementation Bootstraps Web Services. Linux Magazine (2002) `http://www.linux-mag.com/2002-08/axis_01.html`.

5. Buyya, R., ed.: High Performance Cluster Computing, Programming and Applications. Prentice Hall (1999)
6. Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface Standard. (1997)
7. Jones, M., Berkley, C., Bojilova, J., Schildhauer, M.: Managing Scientific Metadata. IEEE Internet Computing **5** (2001) 59–68
8. Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., Weatherspoon, H., Kubiatowicz, J.: Maintenance-Free Global Data Storage. IEEE Internet Computing **5** (2001) 40–49
9. EU-DataGrid (EDG): The DataGrid Project (2003) `http://eu-datagrid.web.cern.ch/eu-datagrid`.
10. Globus: Fundamental Technologies Needed to Build Computational Grids (2003) `http://www.globus.org`.
11. GridLab: A Grid Application Toolkit and Testbed (2003) `http://www.gridlab.org`.
12. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems – Concepts and Design. Addison-Wesley (2001)
13. Fagg, G.E., Gabriel, E., Resch, M., Dongarra, J.J.: Parallel IO Support for Meta-computing Applications: MPI_Connect IO Applied to PACX-MPI. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2131 in Lecture Notes in Computer Science, Springer-Verlag (2001) 135–147 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23-26, 2001, Proceedings.
14. Tsujita, Y., Imamura, T., Takemiya, H., Yamagishi, N.: Stampi-I/O: A Flexible Parallel-I/O Library for Heterogeneous Computing Environment. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2474 in Lecture Notes in Computer Science, Springer-Verlag (2002) 288–295 9th European PVM/MPI Users' Group Meeting, Linz, Austria, September/October, 2002, Proceedings.
15. Sunderam, V., Moyer, S.: PIOUS for PVM (1995) `http://www.mathcs.emory.edu/pious`.
16. Vahdat, A.M., Eastham, P.C., Anderson, T.E.: WebFS: A Global Cache Coherent File System. Technical report, Computer Science Division, University of California Berkeley (1996) `http://www.cs.duke.edu/~vahdat/webfs/webfs.html`.
17. Alexandrov, A.D., Ibel, M., Schauser, K.E., Scheiman, C.J.: Extending the Operating System at the User Level: the Ufo Global File System. In: Proceedings of the USENIX Annual Technical Conference, Anaheim, California, USA (1997) 77–90
18. Bester, J., Foster, I., Kesselman, C., Tedesco, J., Tuecke, S.: GASS: A Data Movement and Access Service for Wide Area Computing Systems. In: Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems, Atlanta, GA, ACM Press (1999) 78–88
19. McClanahan, C.R.: Tomcat: Application Developer's Guide. (2002) Apache Jakarta Project, `http://jakarta.apache.org/tomcat/tomcat-4.1-doc/appdev/index.html`.
20. Czarnul, P.: Programming, Tuning and Automatic Parallelization of Irregular Divide-and-Conquer Applications in DAMPVM/DAC. International Journal of High Performance Computing Applications **17** (2003) 77–93
21. Czarnul, P., Tomko, K., Krawczyk, H.: Dynamic Partitioning of the Divide-and-Conquer Scheme with Migration in PVM Environment. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2131 in Lecture Notes in Computer Science, Springer-Verlag (2001) 174–182 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23-26, 2001, Proceedings.
22. Czarnul, P., Krawczyk, H.: Dynamic Assignment with Process Migration in Distributed Environments. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 1697 in Lecture Notes in Computer Science (1999) 509–516