ViPIOS Islands: Utilizing I/O resources on distributed clusters

Erich Schikuta and Thomas Fuerle
Institut für Informatik und Wirtschaftsinformatik
University of Vienna
Rathausstr. 19/9, A-1010 Vienna, Austria

Abstract

Grid Computing aims to harness resources available on a widely distributed computing infrastructure to enable next generation collaboration frameworks and solution approaches to a stimulating domain of new challenges.

In this paper we present ViPIOS (Vienna Parallel Input Output System) islands, which exploit I/O resources available on the Grid for high performance applications. ViPIOS is a client-server based system tailored for cluster type systems to increase the bandwidth of disk accesses by (re-)distributing the data among available I/O resources and parallelizing the execution scheme.

Secondly we introduce the xDGDL language, a framework to express semantics for data stored on the Grid.

1 Introduction

Two factors strongly influenced the research in high performance computing in the last few years, the I/O bottleneck and cluster systems. Firstly, for many supercomputing applications the limiting factor is not the number of available CPUs anymore, but the bandwidth of the disk I/O system. Secondly, a shift from the classical, costly supercomputer systems to affordable clusters of workstations is apparent, which allows problem solutions to a much lower price.

In the last few years Grid computing became very popular. Approaches like Globus [1], NetSolve, SETI@home attracted more and more people to join. The basic idea behind these projects is to solve large problems by harnessing the CPU cycles of participating machines over the internet. This approach is followed in the small by so called Beowulf cluster type systems. Off-the-shelf workstations are connected by an affordable network interconnect (FastEthernet, Giganet), and suitable operating and programming environments allow to exploit the cumulative processing power to solve grand challenging problems. Due to their low price (compared to the classic supercomputers) these clusters became very popular and representatives can now even be found in the list of the 500 worlds most powerful computer systems (`http://www.top500.org`).

This situation stimulated the development of the Vienna Parallel Input Output System (ViPIOS), which represents a fully-fledged parallel I/O runtime system focusing on cluster systems available on the Grid. An introduction to ViPIOS can be found in [5].

This paper is an extension of [4] focusing onto the Grid and its enabling infrastructure.

2 ViPIOS Islands: Extending ViPIOS for I/O on distributed clusters

2.1 Introduction

The basic concepts of ViPIOS described thus far need some extensions in order to harness I/O resources distributed over the internet. The main challenges in this context are

- The message protocol uses broadcasts in some situations. Since it is clearly impossible to broadcast across the internet some *notion of locality* is needed, which ensures that broadcast messages only have to be sent to a (small) well defined subset of all the ViPIOS server processes running.
- *Name spaces* have to be provided to avoid file naming conflicts.
- *Client grouping* ensures that collaborating client processes can use shared filepointers or access a file exclusively (i.e. only processes belonging to a specific group can use the file concurrently, whereas all other processes are denied access).

- Hard- and software environments across the Internet are very inhomogenous. Hence the *adaptability* of ViPIOS is a major issue. Administrators should be able to tailor the system to their needs.
- Users accessing I/O resources over the Internet generally are unable to overcome errors and faults on the server side (for instance they do not have the rights to restart the server process if it crashes). Therefore some basic *automatic failure recovery* has to be implemented in order to increase the availability of ViPIOS services.
- To make persistent data accessible for a wide range of possible users it is generally not sufficient to just store the data alone. *Meta information* like for instance the datatypes and file formats used has to be supplied too. This is not only valuable for human users it is also vital to enable automatic post processing of the data using OLAP and data warehousing techniques.

2.2 The ViPIOS Island

A *ViPIOS island* is defined to be a closed system with its own name space consisting of a number of ViPIOS servers and a connection controller, which assigns application processes to their buddy servers (a ViPIOS server responsible for the respective connection) on request.

The idea is to segment the distributed I/O services into domains (islands). To reach such an island the client needs to know the hostname (or IP-address) of the connection controller responsible for that island.

2.2.1 The Connection Controller

At any given time, a client or a group of clients can connect/disconnect to/from a ViPIOS island. To connect the client calls an interface function and specifies the hostname (or IP-address) of the targeted island's connection controller. The ViPIOS interface then sends a connect message to that connection controller, which in turn selects a buddy server for the client process (based on information about network topology, data layout and so on). The address of the buddy server is sent back to the ViPIOS interface. The interface converts this address into a *buddy handle* and returns this handle to the calling client process. The client has to use this handle for further requests to the respective ViPIOS island.

A client process may connect to an arbitrary number of ViPIOS islands concurrently (like indicated in Figure 1). Since there is a different buddy server to the application in each island the many-to-one relationship between applications and buddy server holds no more. Each application has exactly one buddy server in each island it is connected to.

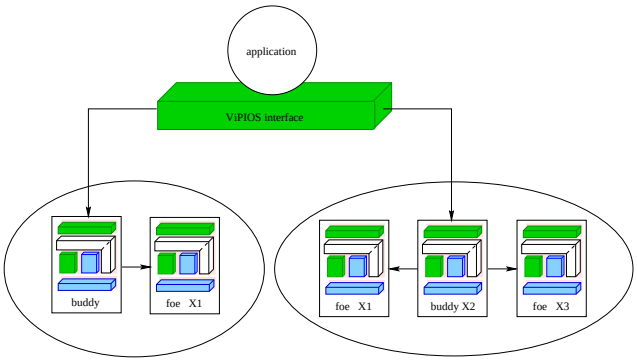

Figure 1: ViPIOS islands

With the standardization of OGSA (Open Grid Service Architecture) [3] we will change our connection model to the standardized service architecture of OGSA.

2.2.2 Name Space of ViPIOS

Each ViPIOS island has its own name space, i.e a file name is unique within an island, but on the other hand the same file name can occur in different islands.

All parts of a single file are stored on one dedicated island. Therefore it is not possible that for any file some bytes have to be retrieved from one island and other bytes have to be retrieved from another island. If a part of the file is located on an island, the rest can be found on the same island. This simple rule restricts the range of broadcast messages to one single island. Whenever a server process searches a part of a file, which can neither be found locally nor by the directory controller, it suffices to broadcast the request to all the other servers on the island. One of them has to hold the data.

To distinguish between files on different islands with the same name, the buddy handle must be specified when opening a file. The call to the open function returns a file handle, which is used by the application to identify the file in all further I/O function calls.

We are working on integrating the name spaces of ViPIOS with the file name lookup functionality of the Datagrid architecture, which can free us from restricting a logical file to a single island.

2.3 Shared File Pointers and Exclusive Access

The decentralized way ViPIOS handles I/O requests minimizes synchronization overhead but poses some problems for operations, which implicitly need some knowledge about the global context. Assume for instance a situation, where two applications with different buddy servers try to access a file exclusively. The two requests to open the file are sent to different servers but only one request may be successful. The other must be rejected in order to guarantee exclusive access. So the servers must somehow find out that there are multiple exclusive requests and resolve the situation.

A similar difficulty arises with shared file pointers. The current state of the file pointer must be stored in some central position, which can be accessed by all the different server processes receiving requests for that file.

To overcome all these trouble each file is assigned a specific ViPIOS server process which is called the *sync controller* of that file. Each file has exactly one sync controller but a sync controller can serve multiple files. Generally the sync controller is chosen to be the same server process that is also the directory controller for that file. If no directory controller exists for the file then the sync controller is the server process, which holds the first byte of the file on its local disks. (Even if the file is empty the distribution strategy chosen by the fragmenter at file creation determines the server which will hold the fist byte of the file and thus the sync controller.)

Now each open request has to contact the sync controller of the file to verify that there are no access conflicts. The current state of a shared file pointer is stored on the sync controller of the file and is thus available to all the servers in the system.

2.4 Group tagging

In parallel computation it is quite common that a number of application processes collaborate to complete a certain task. Under that perspective exclusive access means that only processes belonging to that specific group may access the file but no other processes. Since application processes are executed independently they connect to the ViPIOS system at different points in time and there is no way for ViPIOS to find out, which processes belong together in a group. Each application process therefore must specify the group it belongs to when it connects to a ViPIOS island.

This is done by specifying two additional parameters in the call to the connect function.

- **A user defined group tag.** The application programmer defines a custom group tag, which is a name unique for the ViPIOS island to which the application connects. All the application processes using the same group tag are considered to be members of that group. It is the responsibility of the application programmer to avoid name clashes with other application groups on the same island. This can for example be done by using a GUID as the group tag. Note that the range of a group tag is only a single ViPIOS island. The same tag may be used for different islands producing different and independent groups. Furthermore an application process can connect to different groups on different ViPIOS islands, though it only can be a member of a single group on a specific ViPIOS island at any point in time.

- **The number of group members.** To assure correct handling of access rights the number of the members in the group has also to be specified. Imagine two application processes building a group and having exclusive access to a file. Clearly access for other applications can only be granted after both processes have closed the file. If the processes are not or only loosely synchronized it can happen that the first one already closes the file before the second one even has opened it. In that case the ViPIOS system has to know that there will be a second process that also belongs to the group and will access the file. Or else closing the file would allow other applications to access the file before the group has completed all its file operations.

  To know the number of group members in advance also facilitates some of the optimization tasks of the ViPIOS system (like assigning the best buddy server to each application process or finding the data distribution for a specific file).

2.5 Customizing the System

ViPIOS offers the adjustment of system parameters (like sizes of buffers, number of server processes etc.) to the system administrator who can set these parameters in external configuration files.

These files are interpreted by ViPIOS in a hierarchical manner. A *global configuration file* is used to specify the defaults for all the server processes of a

ViPIOS island. For specific servers these values can be overridden in the *local configuration file* of that server.

If any parameter can not be found (because both of the files are missing or there is no entry in either of the files) ViPIOS uses some predefined parameter values, which are hard coded into the system.

We are also thinking of using a LDAP server for these parameters.

2.6 Failure Recovery

The aim of the failure recovery component of ViPIOS is to provide the stability needed to ensure the availability of the I/O services in a distributed environment. Users accessing the system remotely can not kill or restart server processes that have failed for any reason. In this context there is no intent to recover from hardware failures like a head crash on the hard disk or something similar severe. But the system is designed to survive minor failures like temporary unavailability of servers, network congestion, buffer overrun or memory exhaustion.

2.6.1 Spawning of Server Processes

The connection controller plays the major role in failure recovery. It uses periodic keep alive requests to ensure that all servers on the island are still running. If any of the servers has terminated unexpectedly, the connection controller tries to restart it. If the restart fails some files may become inaccessible (i.e. the files local to the server process, which can not be restarted). The applications are informed of that fact and open requests to those files are canceled gracefully.

The connection controller itself is monitored by a watchdog process, which will restart it immediately, if necessary.

2.7 Unified Messaging

ViPIOS is based on an unified messaging system, which allows for changing the underlying messaging system like PVM and MPI.

The main focus for this development was the "best bread" approach of combining basically the dynamic features of PVM with the data structure features of MPI transparent to the programmer, which we need for the design of our internal APIs and external interfaces (see refsec interfaces).

3 Interfaces

ViPIOS offers a wide variety of (external) interfaces for different purposes.

The main interfaces are:

- *A native ViPIOS interface*, which is functionally viewed a superset of the traditional Unix interface, with extensions similar to MPI-IO and PVFS. It is used internally, but can also be used for application programming.
- *ViMPIOS: a MPI-IO interface*, which is an almost complete implementation of chapter 9 of the MPI2 draft.
- *ViPFS: a file system interface*, which implements a file system with its common tools on top of ViPIOS delivering persistence and a canonical view for the distributed files.

A novel approach in ViPIOS is the introduction of a XML based language to express data semantics within the stated interfaces.

4 Data Semantics

New and stimulating problems in biology, physics, etc. arise, which bring new high performance applications with the need to store, administer and search intelligently gigantic data set distributed over local and global storage medias.

We will face a similar situation as in the well-known area of database systems, where data represents a model of the reality. It can be searched, analyzed, easy administered and is efficiently at hand for arbitrary applications. This makes necessary means to express semantics in the data and consequently the need for mechanisms expressing semantics. Data has to be attributed with meta information describing the specific semantics of the information in a standardized and processable way. This meta data allows applications to search the stored information intelligently.

However meta information in the context of high performance applications has to describe not only the logical knowledge within the data (semantic information) but also specific structural problem information of the parallel and/or distributed execution (syntactical information). Thus we developed a XML based language, xDGDL (XML Data Grid Description Language), which provides a homogenous framework for describing data on all interpretative levels (from physical representation to logical information). In our

framework physical files are consequently augmented by interpretative information specified in the proposed language.

To allow full flexibility for the programmer and the administration methods, we propose an architecture with three independent layers in the parallel I/O architecture (similar to the three-level-architecture of database systems):

- Problem layer. Defines the problem specific data distribution among the cooperating parallel processes.
- File layer. Provides a composed (canonical) view of the persistently stored data in the system.
- Data layer. Defines the physical data distribution among the available disks.

Consequently it is our goal to propose a framework, which allows to express the information on all layers of the presented file hierarchy. This led to the design of the xDGDL, which acts in the system in two ways; on one hand it provides a user interface allowing to specify the layout of the file, on the other hand it is the expressive mechanism within the system to administer the distribution information of the files stored in the file system across several sites on the Grid.

Due to paper limitation we can only give a short description of xDGDL. A typical xDGDL description consists of the following elements:

- **Document Root** The root of the document specifies the version and timestamp of the file of the XML description.
- **Island** Defines a logical unit with several servers distributed worldwide.
- **Servers** Servers are physical machines identified by their host name.
- **Devices** Devices are the disks holding the data on the specific server.
- **View** The View element allows a specific distribution on the available devices.
- **Block** The Block element specifies the number of bytes to write to the specific disk.

4.0.1 Document Root.

The root of the document is described by the element `PARSTORAGE`. The root element can contain several child elements:

- `PROCESSORS` describes the named processor arrays.
- `TYPE` describes the data types and variables stored in the logical file.
- `ALIGN` describes the alignments of the variables.
- `ISLAND` describes the physical view of the file.

4.0.2 Island.

The `ISLAND` describes several servers interconnected. These servers can be distributed across the Grid. The island is identified by an island name. The `ISLAND` consists of one or more servers.

4.0.3 Servers.

The `SERVER` element is identified uniquely on a node. It has an attribute called `HOST` which mirrors the domain name of the server.

The `SERVER` element consists of one or more `DEVICE` elements.

4.0.4 Devices.

Devices are the disks holding the data on the specific server. The devices need not be physical, even a mounted NFS device on another server can be a device which can be accessed from a ViPIOS server. To describe the structure of file parts to be written to disk, a `VIEW` is used.

4.0.5 View.

The `VIEW` element is the link between logical, physical and application view. It is responsible for transforming the internal structure of the data layout to application programs. The `VIEW` element consists of one or more `BLOCK` elements.

4.0.6 Block.

The `BLOCK` element can have two types of child nodes. It can have a `BYTEBLOCK` element, which means, that either there are no more `VIEW` elements or it can consist of `VIEW` elements which have one or more `BLOCK` elements themselves. This leads to a recursive structure which allows arbitrary distribution. The `BLOCK` element consists of the following attributes:

- `OFFSET` describes how many bytes should be skipped from the starting point of the current `BLOCK`.

- `REPEAT` describes how often the `BLOCK` should be read/written.
- `COUNT` number of bytes to read/write at each `BLOCK` operation.
- `STRIDE` describes the number of bytes to skip at each `BLOCK` operation.

As an example, a file distributed onto two-server within ViPIOS can be described by the following xDGDL specification:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE PARSTORAGE SYSTEM "XDGDL.dtd">
<PARSTORAGE VERSION="1.0"
 TIMESTAMP="testfile_regular">
<ISLAND NAME="island1.pri.univie.ac.at">
 <SERVER HOST="vipios1.pri.univie.ac.at">
  <DEVICE DEVICE_ID="/dev/vda1">
   <VIEW SKIP_HEADER="0" SKIP="7">
    <BLOCK OFFSET="0" REPEAT="3"
           COUNT="5" STRIDE="7">
     <BYTEBLOCK/>
    </BLOCK>
   </VIEW>
  </DEVICE>
 </SERVER>
 <SERVER HOST="vipios2.pri.univie.ac.at">
  <DEVICE DEVICE_ID="/dev/vda1">
   <VIEW SKIP_HEADER="0" SKIP="0">
    <BLOCK OFFSET="5" REPEAT="3"
           COUNT="7" STRIDE="5">
     <BYTEBLOCK/>
    </BLOCK>
   </VIEW>
  </DEVICE>
 </SERVER>
</ISLAND> </PARSTORAGE>
```

More in-depth information on the xDGDL can be found in [2].

5 Conclusions and future work

We presented ViPIOS and its extensions for addressing the needs of distributed I/O on the Grid. Work on a final implementation of the ViPIOS island is on the way. Due to the challenging situation of the the Grid research with every day new proposal for standards we are thoroughly evaluating the different approaches and are eager to integrate novel Grid standards into ViPIOS.

Further we introduced in this paper xDGDL, a XML language for storing meta information for distributed files, which allows to express semantic information on various levels. This language is our contribution to the standardization process on the Grid. We think that the semantic description of data enables new ways administrating, searching, analyzing and computing information on the Grid.

Acknowledgement

On this occasion we thank Andras Belokosztolszki, Rene Felder and Helmut Wanek, who helped on various aspect developing ViPIOS. The work described in this paper was partly supported by the Special Research Program SFB F011 AURORA of the Austrian Science Fund.

References

[1] Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, and Steven Tuecke. 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*, pages 78–88, Atlanta, GA, May 1999. ACM Press.

[2] Rene Felder and Erich Schikuta. Towards a XML based data grid description language. submitted for publication.

[3] Ian Foster, Carl Kesselman, Jefferey Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. http://www.globus.org/research/papers/ogsa.pdf, 2002.

[4] Thomas Fuerle, Oliver Jorns, Erich Schikuta, and Helmut Wanek. Meta-ViPIOS: Harness distributed i/o resources with vipios. *Iberoamerican Journal of Research "Computing and Systems", Special Issue on Parallel Computing*, 4(2):124–142, December 2000.

[5] Erich Schikuta, Thomas Fuerle, and Helmut Wanek. ViPIOS: The Vienna Parallel Input/Output System. In *Proc. of the Euro-Par'98*, Lecture Notes in Computer Science, Southampton, England, September 1998. Springer-Verlag.