

HDF Support For Network Of Workstations

White Paper

April 1996

Prepared Under NRA#???

(U. of Illinois Ref. No. ???)

Albert Cheng

Michael Folk

Ruey-Hsia Li

National Center for Supercomputing Applications

University of Illinois at Urbana-Champaign

Table of Contents

1. INTRODUCTION	1
1.1 Overview	1
1.2 Design of the original HDF	1
2. NEW CHALLENGE OF PARALLEL COMPUTING	3
2.1 Advances in Parallel Computing	3
2.2 Kinds of Parallel Computers	3
2.3 Parallel programming models	3
2.4 Architecture of NOW	4
2.5 Needs of HDF for NOW	4
3. DESIGN ISSUES OF HDF FOR NOW	9
3.1 Deficiencies of the HDF library	9
3.1.1 No concurrent processing	9
3.1.2 No parallel input/output	9
3.1.3 No parallel storage	9
3.2 Distributed memory	9
3.3 Distributed filesystems	10
3.4 Platform homogeneity	10
4. DISTRIBUTED HDF FILE	11
4.1 Single HDF Server	11
4.2 Multiple IO Channels	11
4.3 Multiple HDF IO	12
5. MOVING OBJECTS BETWEEN MEMORY AND FILES	18
5.1 Process roles	18
5.2 Central hub model	18
5.3 Switch board model	19

TABLE OF FIGURES	
FIGURE 1-1 SINGLE PROCESSOR MACHINE	2
FIGURE 2-1 SHARED MEMORY AND SINGLE FILESYSTEM MACHINE	5
FIGURE 2-2 DISTRIBUTED MEMORY AND SINGLE FILESYSTEM MACHINE	6
FIGURE 2-3 DISTRIBUTED MEMORY AND DISTRIBUTED FILESYSTEM MACHINE	7
FIGURE 2-4 ARCHITECTURE OF NOW	8
FIGURE 4-1 SINGLE HDF SERVER	13
FIGURE 4-2 MULTIPLE IO CHANNEL	14
FIGURE 4-3 HDF OBJECT DISTRIBUTED OVER MULTIPLE HDF FILES	15
FIGURE 4-4 MULTIPLE HDF IO	16
FIGURE 4-5 DISTRIBUTED HDF FOR NOW	17
FIGURE 5-1 CENTRAL HUB MODEL	20
FIGURE 5-2 SWITCH BOARD MODEL	21

1. Introduction

1.1 Overview

The *Hierarchical Data Format (HDF)* is a multi-object file format for sharing scientific data between different computing platforms. It is designed for the traditional sequential computers and has been well accepted in the scientific communities. But given the advance of parallel computers in the nineties, HDF, as is, is not adequate in the parallel computing environments. This paper describes how HDF can be extended to support one type of parallel environment, the Network of Workstations (**NOW**). The rest of this section describes briefly how the current version of HDF was designed.

Section 2 describes the range of parallel computers and parallel computing models.

Section 3 describes issues concerning the design of HDF for NOW.

Section 4 describes different possible extensions to the HDF file models and their advantages and disadvantages in supporting NOW.

Section 5 describes different programming models in moving objects between memory and HDF files.

Section 6 presents a preliminary design of an *HDF/NOW* programming interface.

Section 7 presents potential additions and improvements for the HDF/NOW being proposed.

1.2 Design of the original HDF

HDF was created at the National Center for Supercomputing Applications (NCSA) in 1987 to support data storage and exchange for the scientific community. It was designed for the traditional sequential flow control computers, as those were the commonly available machines (Figure 1-1). Its design assumes

- single process access of objects;
- all objects are in one single memory space;
- all objects are wholly stored in a single file;
- sequential input/output systems.

The design was first implemented in UNIX systems which were mainly sequential. HDF was later implemented for the MacOS and DOS systems and was adequate for all the platforms HDF supported back then. Parallel computers did exist then but they were high cost special purposes machines that were not commonly accessible to the mass. The supercomputers at NCSA were multi-processors parallel computers (e.g. Cray X-MP, Y-MP, Convex) but operating systems all supported the single memory space and single filesystem that they appeared as sequential machines to the users. The HDF library code was improved to make use of the vector processing in those machines, but HDF I/O remained sequential.

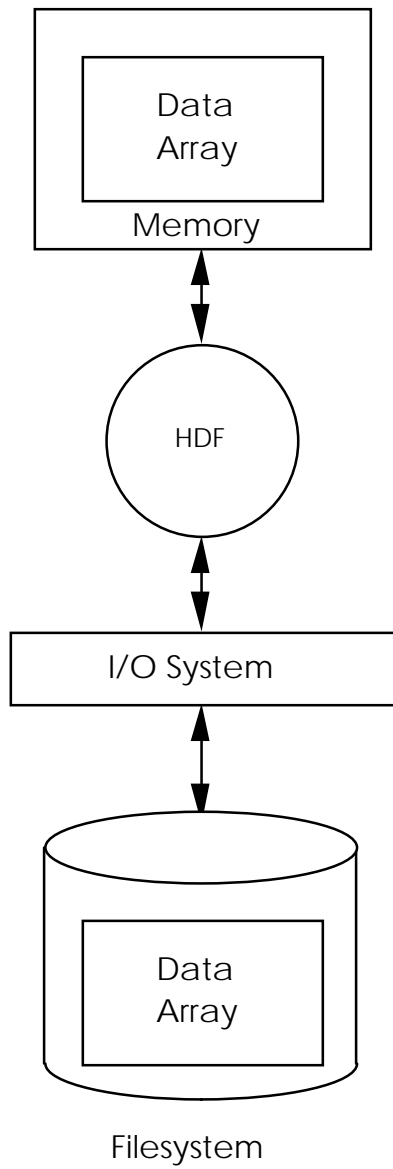


Figure 1-1 Single Processor Machine

2. New Challenge of Parallel Computing

This section describes different kinds of parallel computers and parallel computing models. It then describes the characteristics of NOW.

2.1 Advances in Parallel Computing

Advances in hardware technology and the software development makes parallel computing feasible. On the high end, they can provide more raw computing cycles and memory space for grand challenge problems. On the opposite end, they provide a scalable computing system that consists of common hardware parts and can be accessed in various configuration according to computing needs.

2.2 Kinds of Parallel Computers

There are many ways to classify parallel computers but one way relevant to the support of HDF is by the difference in memory space and filesystems. Memory space can be shared or distributed, and the filesystem can be single or distributed.

In shared memory machines, all processors share the same memory space, though not necessarily with equal access speed to all memory. In the distributed memory machines, each processor has direct access to a portion of memory local to it. To access memory space not local to a processor, it has to communicate with the processor that “possesses” that portion of memory.

In the single filesystem machines, all processors share a single filesystem. This can be achieved by a common network layer between the processors and the storage disks (e.g. CM5, a Convex Exemplar cluster). In distributed filesystem machines, each processor has its own local filesystem and if it needs to access file data remote from it, it has to communicate with the processor that holds that file.

With different combinations of memory space and filesystems, there can be three groups of parallel computers,

- Shared memory and single filesystem (Figure 2-1)
E.g., Cray Y-MP, SGI Power Challenge clusters
- Distributed memory and single filesystem (Figure 2-2)
E.g., CM5
- Distributed memory and distributed filesystem (Figure 2-3)
E.g., IBM SP-2

We are not aware of machines that have shared memory but distributed filesystems.

2.3 Parallel programming models

There are two coarse ways to differentiate programming models in parallel systems. They are SIMD (Same Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data). SIMD is commonly employed in shared memory parallel computers and MIMD is for distributed memory systems. However, it is possible to use either models in either

memory space types. For example, the CM5, though a distributed memory machine, provides a parallel FORTRAN language (CMFORTRAN) that supports the SIMD programming model. The SGI cluster, though a shared memory system, has implemented the Message Passing Interface (MPI) to support the MIMD programming model.

2.4 Architecture of NOW

The Network Of Workstations (NOW) architecture belongs to the distributed memory and distributed filesystem group. It consists of multiple workstations, often of different architectures, networked together to support parallel computing, also known as distributed computing. Each workstation has its own memory and usually a local filesystem (Figure 2-4). (Diskless workstations are not as common as they once were.) A NOW system may employ a network file system (e.g. NFS or AFS) to share common files such as users' home directories, but that does not alter the issues at hand since it is common for each workstation to possess a local scratch filesystem for better IO speed. To take advantage of the different strengths of different workstations in a NOW system, the MIMD programming model is a better fit, though it requires more effort in programming.

2.5 Needs of HDF for NOW

Scientists have moved to the NOW for several reasons,

- Lower cost
- Scalable computing power
- General purpose machines
- Familiar working environment

But NOW users who have used HDF in their work need a data file format like HDF to give them support for

- a standard self-described data format ;
- machine independent data storage;
- a machine independent programming interface;
- efficient parallel I/O for large data objects.

HDF can also provide extra advantages for NOW users:

- data compression for large data objects;
- a format exchangeable with traditional sequential machines;
- visualization tools that recognize the HDF data format.

The Relativity study group at NCSA exemplifies a group of users who need such a data format.

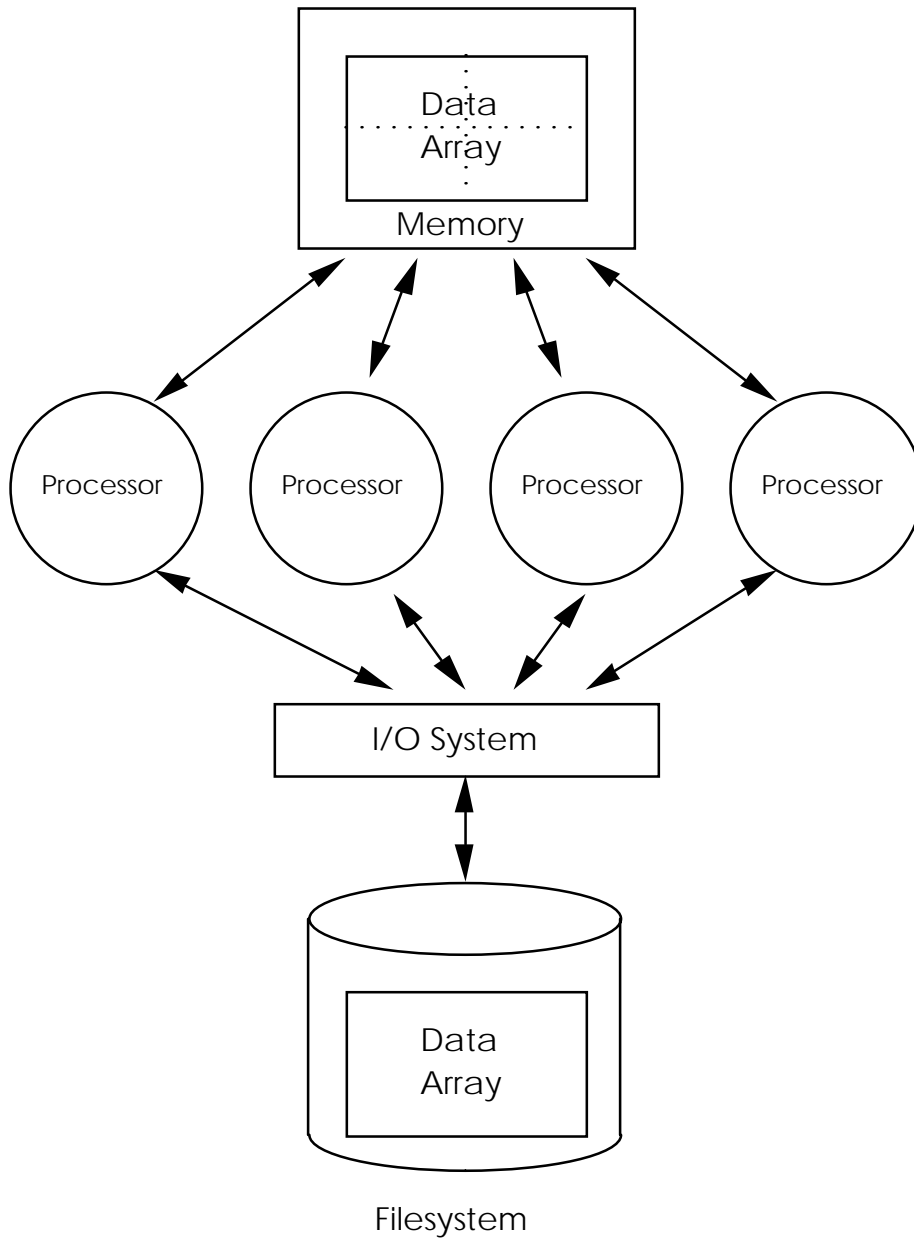


Figure 2-1 Shared Memory And Single Filesystem Machine

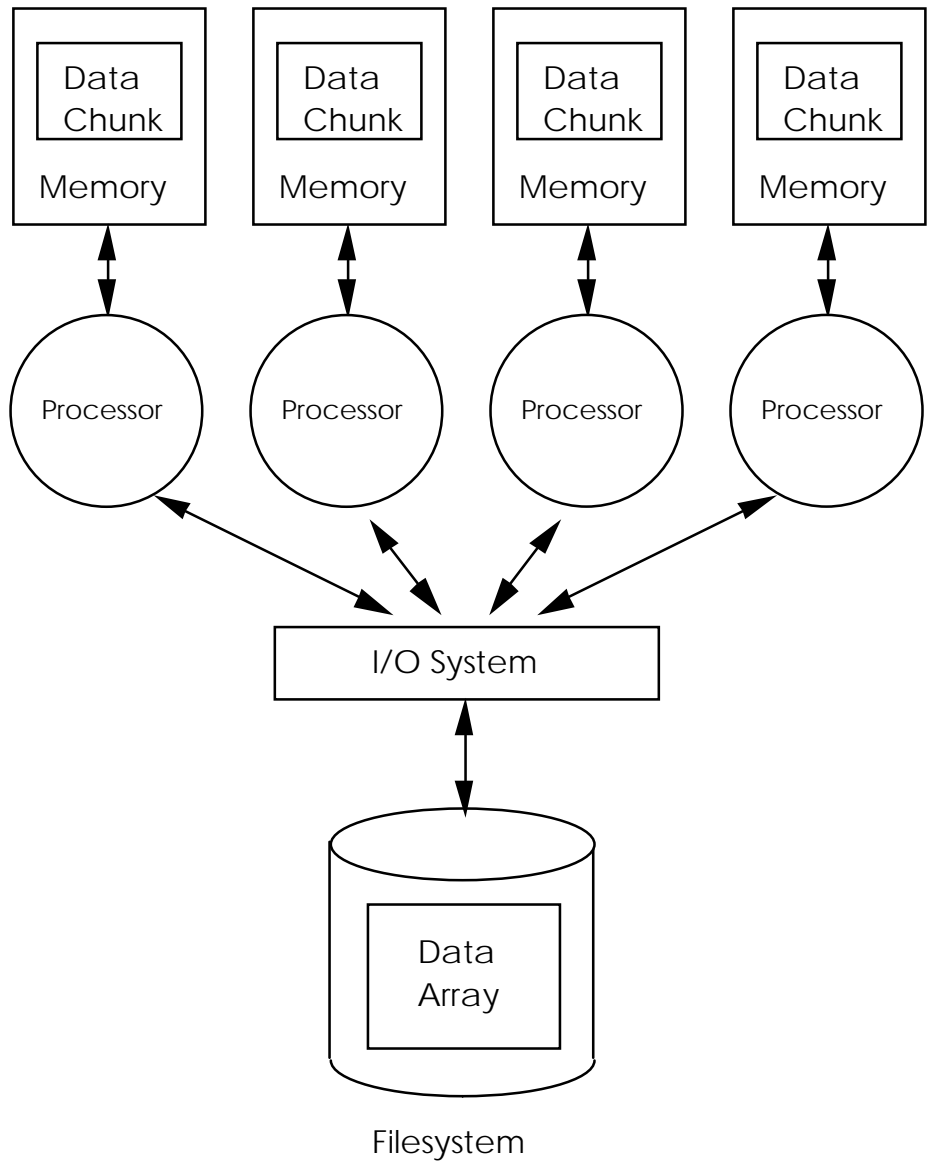


Figure 2-2 Distributed Memory and Single Filesystem Machine

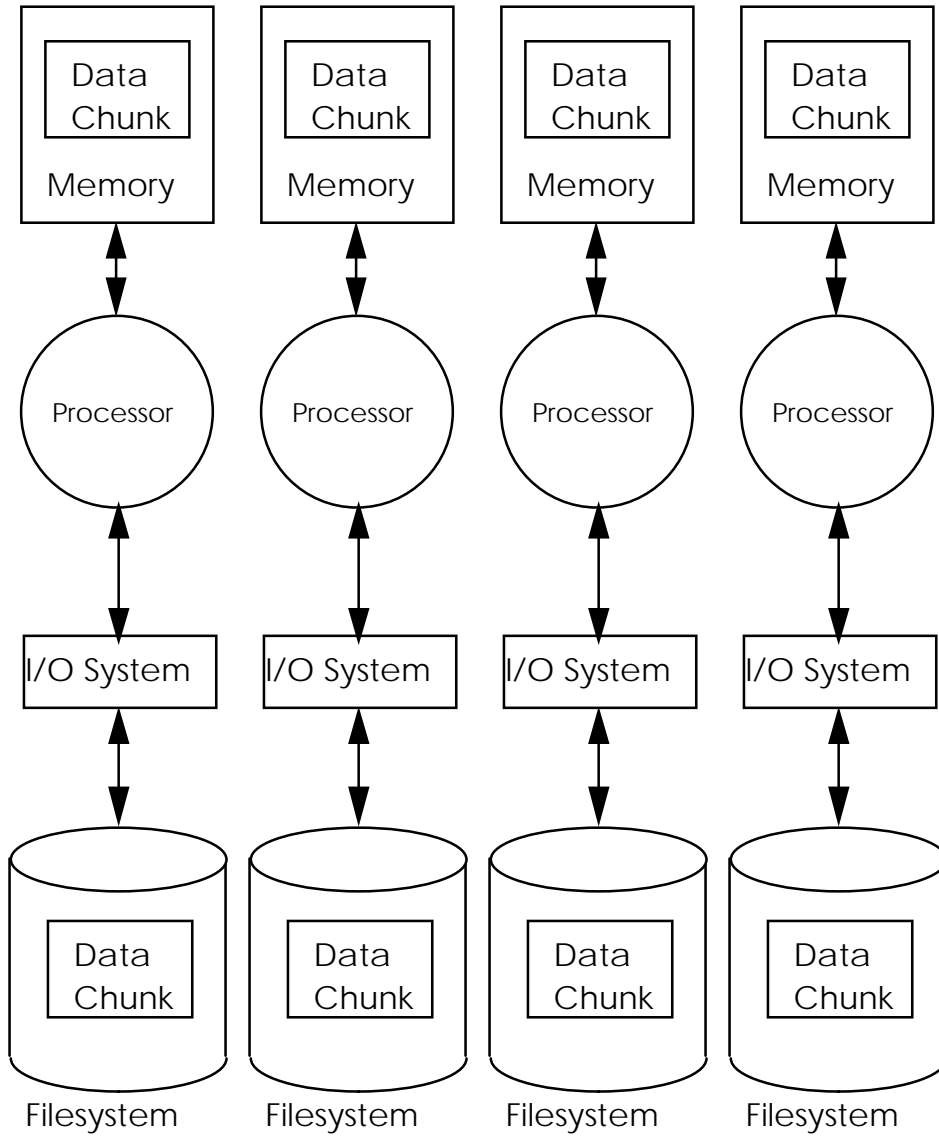


Figure 2-3 Distributed Memory and Distributed Filesystem Machine

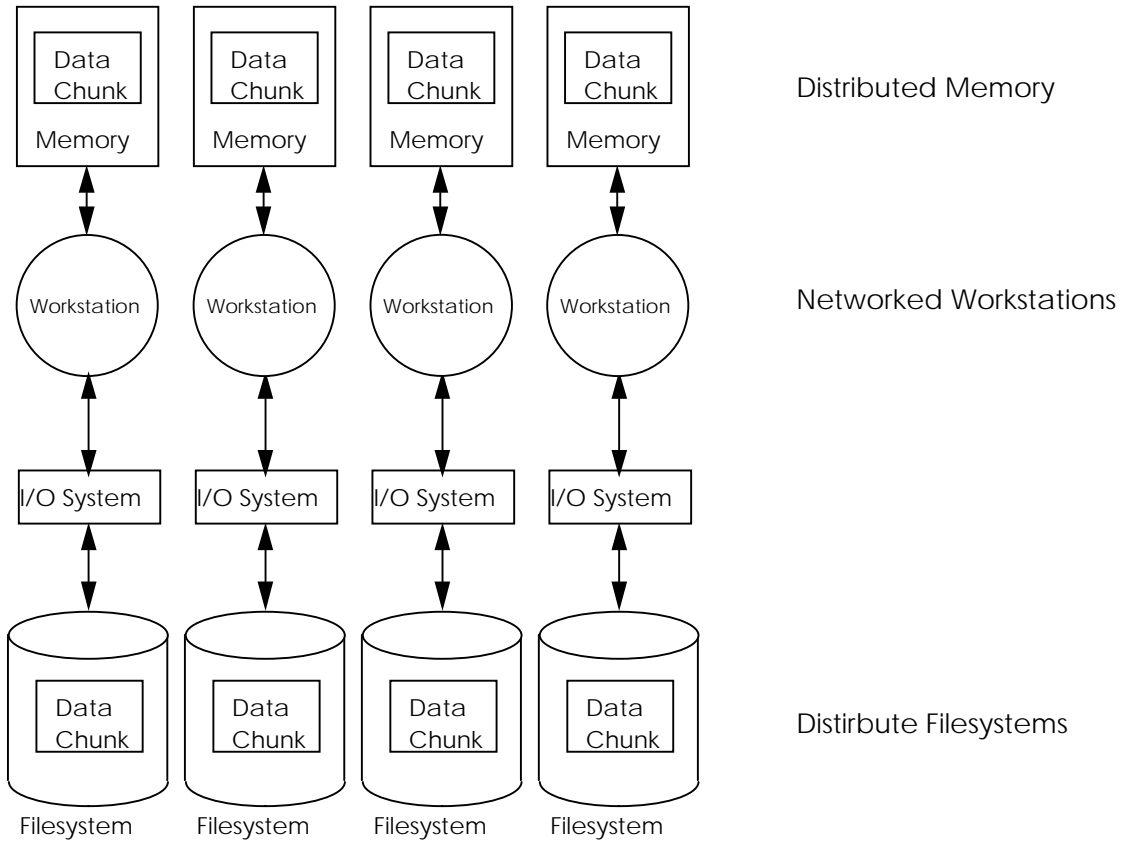


Figure 2-4 Architecture of NOW

3. Design Issues Of HDF For NOW

This section describes the issues concerning the designs of HDF for NOW. We first present the deficiencies of the current HDF library if used in the NOW environment. Then we discuss three issues, *distributed memory*, *distributed filesystem*, and *the platform homogeneity* that are particular important in extending HDF library to support the NOW environment.

3.1 Deficiencies of the HDF library

3.1.1 No concurrent processing

The current HDF library assumes single process access only. An HDF file does not permit concurrent accesses by multiple processes. Also, the library code is not thread safe. This prohibits multi-threads executions common in the NOW environment.

3.1.2 No parallel input/output

The HDF library uses the typical UNIX input/output system calls for the file IO. The parallel IO concept is not considered. (One exception is the CM5 parallel IO extension for external elements.)

3.1.3 No parallel storage

The HDF library assumes that each data object is wholly contained in an HDF file, or as an external element. Even in the case of external elements, HDF assumes each external element is wholly contained in an external file. So, having an HDF file distributed across a distributed filesystem of a NOW system is not possible.

3.2 Distributed memory

In a shared memory system, all participating parallel processes share one memory space. All processes can access all memory locations directly but not necessarily with equal access speed. These systems usually support a shared filesystem such that user applications may view the whole system as a single process system. The CM5 machine provides such a model in its CMFORTRAN and C* programming environments. The Power Challenge and Exemplar clusters are another two examples. By making use of the parallel I/O system for the SDA (Scalable Disk Array), we have implemented an extension to HDF to support parallel I/O for data arrays in the CM5 system at NCSA.

In a distributed memory system, each process can access its own memory directly, but to access data that are in the memory of other processes, two processes have to cooperate by passing data as messages between them. The original design of the HDF library is based on a common memory space. HDF needs to adapt the concept of distributed objects in memory. For example, instead of having a whole data array object in memory, each participating process could hold a chunk of the array. When a process needs to access data that is not of its local chunk, it has to communicate with the other processes that hold the data.

3.3 Distributed filesystems

In a shared filesystem system, all participating processes have direct access to all data files in the system as if all were in one filesystem. This can be supported via hardware as in the case of an Exemplar hypernode in which the eight processors share one common filesystem, or via software such as NFS, which provides network access to the filesystem of individual workstations.

In a distributed filesystem system, each participating process has its own local filesystem. When a process needs to access file data outside of its own filesystem, it has to communicate with another process for data access. One example of this is the SP-2 system, in which each processor node has its local filesystem for faster local I/O. Another example is the Power Challenge cluster or the Exemplar hypernode. Each cluster or hypernode has a shared filesystem but between clusters or hypernodes they share a distributed filesystem.

The original HDF is also designed to assume the HDF file resides wholly in the local filesystem. The *external element* feature of an HDF file allows the storage of some parts of the HDF file outside of the main file, but that still requires that *external elements* reside wholly in the same single filesystem.

The current HDF library assumes exclusive access to an HDF file when it opens the file with write access. The library does not implement any file locking mechanism. This poses a big problem in the parallel computing environment when multiple processes will access the data in parallel. Although ideally the HDF-NOW design needs to support concurrent access to the data object, this feature is not planned for early prototypes of HDF-NOW.

3.4 Platform homogeneity

A NOW environment can be classified as a *Homogeneous* or a *Heterogeneous* system according to the mix of workstations it contains.

A *homogeneous system* consists of workstations of the same platform. In such a system, all processes use the same data representation, making data exchange simple.

A heterogeneous system consists of workstations of different platforms. An example is a system consisting of SGI's and HP's networked together. This provides flexibility in combining resources. For example, one could combine number crunching machines with big disk storage machines to handle a large simulation project. A heterogeneous system also provides the flexibility of using as much resources as are available to solve the task. But it also adds the complexity of mixed data representations and other problems. Since HDF provides the support of automatic data conversion for information exchange among machines of different platforms, supporting a *Heterogeneous* NOW environment is not a problem for HDF.

4. Distributed HDF File

This section describes several designs for supporting HDF for a distributed filesystem in a NOW environment. We will present different physical layouts of the HDF file together with an analysis of the advantages and disadvantages of each design.

4.1 *Single HDF Server*

The HDF file is stored in the filesystem of one workstation which is designated as the Main Workstation (MWS) among all the participating workstations. A process, called HDF Server, will run in the MWS to handle all HDF file accesses (Figure 4-1). When any process needs to access an HDF object, be it read or write, it communicates with the Server which does the disk accesses on behalf of the other processes. In the case of reading in an HDF object, the Server reads in the HDF object from the HDF file and delivers the data to the other processes according to the current defined distributed layout among the processes. In the case of writing an HDF object, the Server gathers the data of the object from all other processes and writes it out to the HDF file.

The advantages for this design is that it is simple and straight forward. It only requires new coding of the Server to coordinate and maintain how the HDF object is distributed among the processes. No change to the HDF file layout is required at all. The HDF file can be moved to any workstation and starts to work without any change. Indeed, some HDF users have reported implementing such a server for their own projects working in distributed memory systems.

The disadvantages for this simple approach are:

- The Server becomes a bottleneck. It can become busy serving all other processes with a great deal of communication traffic, something to avoid in the NOW systems.
- IO speed and file sizes are limited to the MWS capacity.

4.2 *Multiple IO Channels*

The HDF file, called the Main HDF File (MHF), is stored in the Main Workstation and a process is designated as the Main HDF Process (**MHP**) for access to the MHF. The MHF holds all metadata for the HDF objects, but the data portion of the objects is divided into chunks, which are stored as external elements in HDF External Files in the filesystems of other workstations (Figure 4-2). The External File format is basically a binary file of the data stored in the HDF data representation format. In each workstation that contains the External Element files, a process, called Associate HDF Process (AHP), is designated to access the files on behalf of all other processes. In the case of writing an HDF object, the MHP directs all AHP's to gather the data of the objects and to write them out in chunks to each corresponding External file. The MHP keeps track of where the chunks are and stores the information, together with all metadata of the object in the MHF. In the case of reading an HDF object, the MHP reads in the *metadata* and directs the appropriate AHP's to read the data chunks from their External Element files and to send them to the appropriate processes.

The advantages of *Multiple IO Channels* are many:

- It has faster throughput as IO are done in parallel by multiple processes instead of by the MHP only.
- Objects of larger sizes can be stored. A workstation may have a system limit of disk file size lower than the 2 GB size HDF has. Such a limit can be bypassed by storing the data portion in multiple external files across multiple workstations.
- There is much less network traffic between the compute processes and the MHP. The processes can perform I/O in an asynchronous manner rather than queued up in a sequential manner as in the Single HDF Server design.

The disadvantages for this are the following.

- The External Files of the chunks do not contain any identification. They can get lost or misinterpreted by other applications.
- The External Files do not support data compression or other alternative storage schemes.
- If the *HDF group files* are moved to other machines, the user has to gather all “*pieces*” of it. It is error prone and tedious.
- The current HDF library API does not support direct access to the External File. It must go through as an HDF object that happens to be an External Element. New coding in the HDF library kernel code would be required.

4.3 Multiple HDF IO

This is a refinement of the *Multiple IO channels*. Instead of just writing the chunks to External Files, each Associate HDF Process stores them in an HDF file called an Associate HDF File (AHF). The AHP then sends the AHF file name and the chunk object header information to the Main HDF Process which stores the information in the *Main HDF File* (Figure 4-3). In the case of reading, the MHP reads in the header information of the distributed chunk objects. It then directs the appropriate AHPs to read in the data chunks and to send them to the appropriate processes. It works similarly in the case of writing (Figure 4-4).

The advantages of this over the *Multiple IO Channels* are as the following,

- All the Associate HDF Files contain self-describing meta information. This makes it easier to keep track of the chunks and less chance of a chunk being misinterpreted compared to raw binary formatted files.
- One can store multiple data chunks with appropriate attributes in one single file rather than many binary files. (One can, in the design of Multi-IO-Channel, lump all binary files into one big file, but that makes the bookkeeping even harder.)
- The AHF can employ data compression or other alternative storage schemes.

- The Associate HDF Files can be accessed by the current HDF library just like any HDF files.

The main disadvantages of *Multiple HDF IO* is that there is overhead in using an HDF file format. The IO speed will be slower and the files are slightly bigger. But the HDF library code has very good IO speed compare to the native IO speed and the overhead in disk space is a small price to pay for the benefits of a self-described data file.

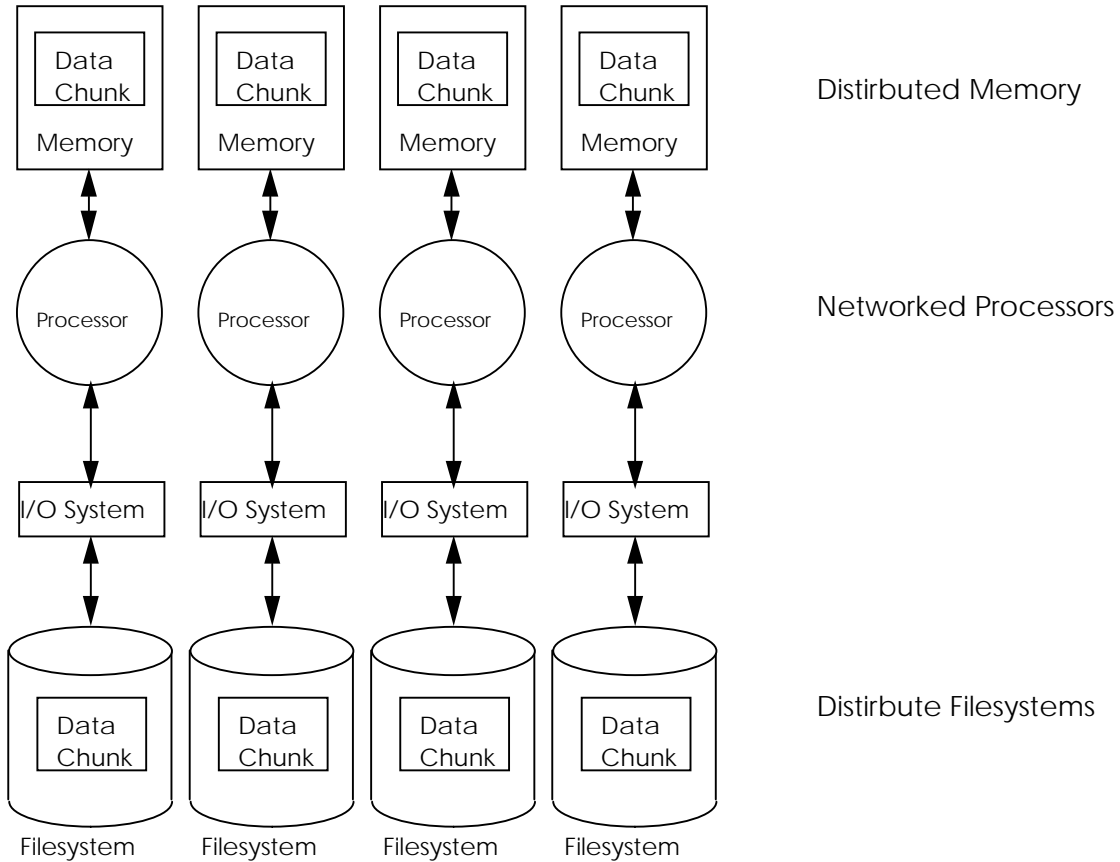


Figure 4-1 Single HDF Server

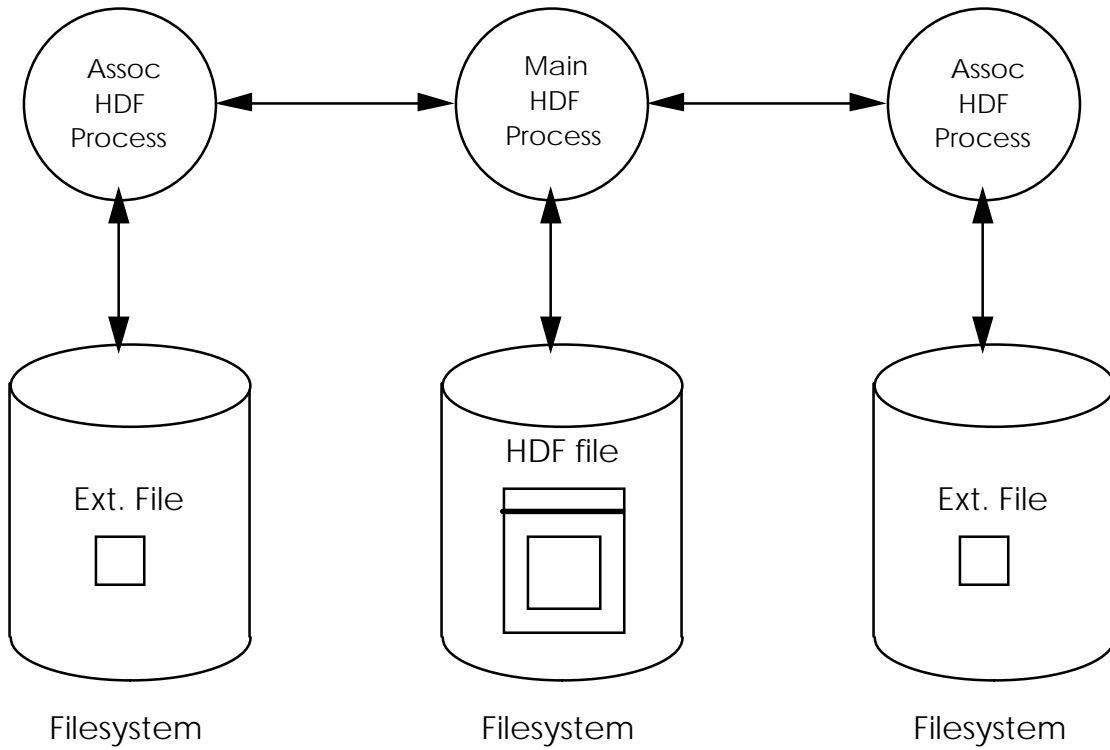


Figure 4-2 Multiple IO Channel

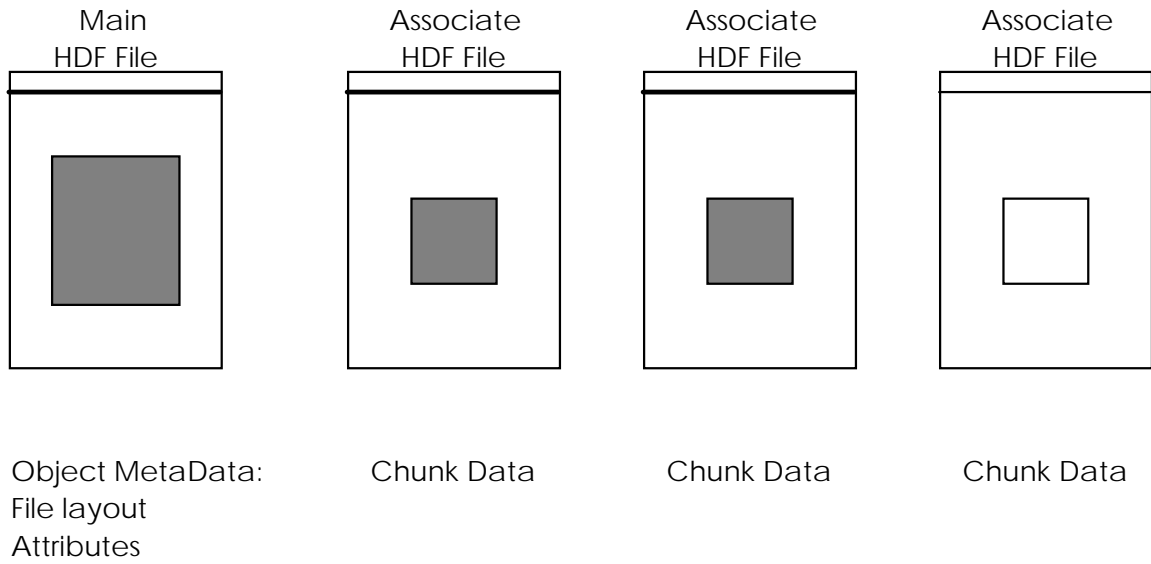


Figure 4-3 HDF Object Distributed Over Multiple HDF Files

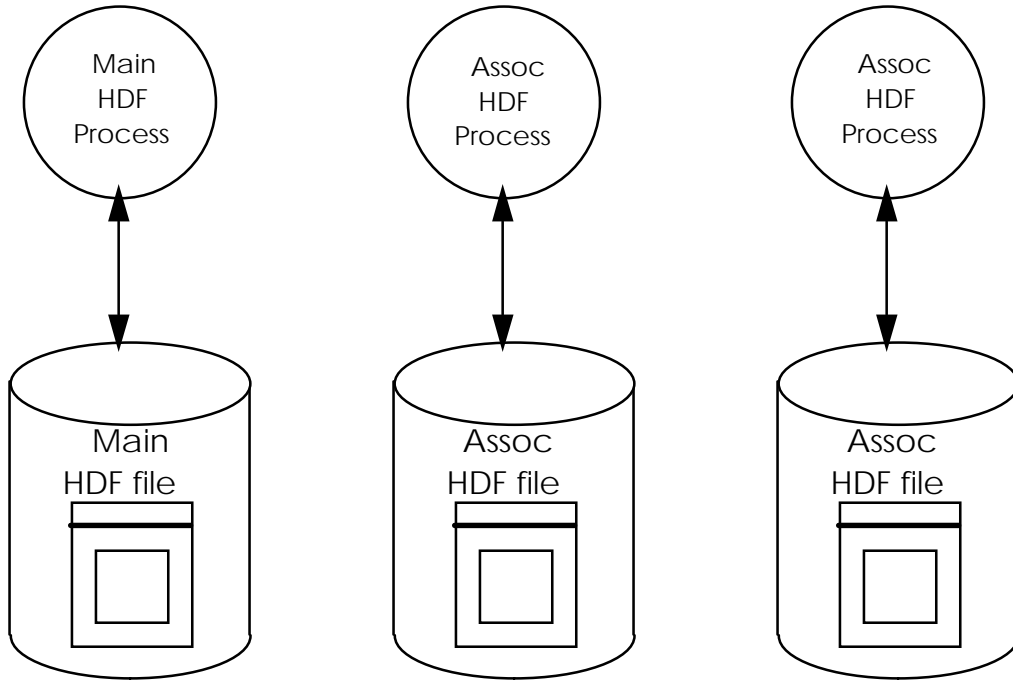


Figure 4-4 Multiple HDF IO

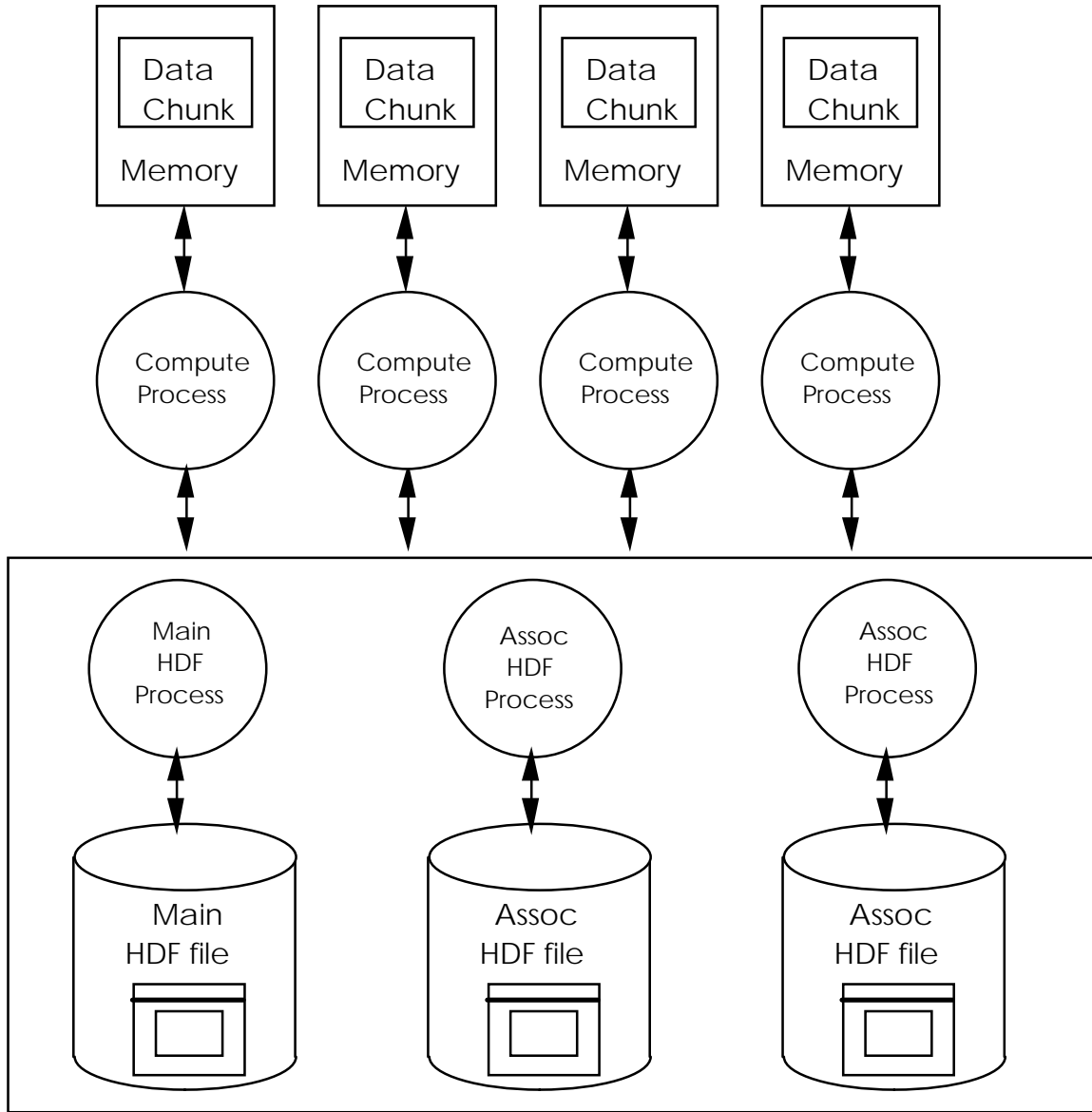


Figure 4-5 Distributed HDF for NOW

5. Moving Objects Between Memory And Files

This section describes aspects of the design dealing with moving HDF objects between the distributed HDF files and the distributed memory in the NOW environment. We present two possible programming models for the users.

5.1 Process roles

In the **Multi-HDF-IO** model proposed in the last section, there are processes designated for I/O processing with the HDF files. They are classified as the *IO Processes*. Other processes in the NOW environment that do computation with the object data are called *Compute Processes*. It is desirable to have the object layout in the distributed memory of the Compute Processes the same as the layout in the HDF files, but in practice that is often not the case. E.g., the file layout for a data array can consist of 4 chunks of blocks but the memory layout consists of 10 slabs (Figure??). Also, HDF files are intended for data exchange between different applications. It is unreasonable to require all applications using the data files to use the same memory layout. Therefore, HDF must support the re-distribution of objects from the file layout to the memory layout in the NOW environment. Two designs are presented next.

5.2 Central hub model

In the *Central hub model*, the MHP acts as a hub through which all messages must pass. The Main HDF Process keeps tracks of where all the data chunks are. Whenever a compute process wants to do IO for a part of an HDF object, it sends the request to the MHP (Figure 5-1). In the case of write, it sends the data to MHP which in turn sends it to the appropriate Associate HDF Process that holds the HDF file containing that part of the object. Sometimes the data to be written may involve HDF files of more than one AHP. In that case, the MHP is responsible for dividing up the data and sending each portion to the appropriate AHP for writing.

In the case of read, the compute process sends the request to the MHP which fetches the data needed from various AHP's, combines the data chunks into one big chunk and send it to the compute process that has requested it.

The advantages of this design is that the MHP knows *everything* about the data organization and all the other processes just use it as the broker for all HDF file accesses. It is very simple for them.

The disadvantages of this design is that there is a need to pass many data messages. Every IO to an HDF file involves at least a message between the compute process and the MHP, plus messages between the MHP and other AHP's. The MHP again may become a bottleneck since requests from different compute processes are processed by the MHP in a sequential manner.

5.3 Switch board model

In the *Switch board model* there is still an MPH that keeps track of all data chunks, but it does not get involve in actual data passing. A compute process asks the MHP *where* the data are. The MHP tells it which AHPs have the data it wants. The compute process then communicates directly with the AHPs to arrange for data exchange (Figure 5-2).

The advantage is that the MHP just gives out locations of data and it can proceed to process requests from other processes while the data exchange occurs between the compute process and the AHP. There is less message traffic since the data is passed directly between the AHP and the compute process.

The disadvantage is that the coding for the compute process is more complex since it has to communicate with the MHP and the AHPs. Despite this, the improvement in network performance is worth the extra coding.

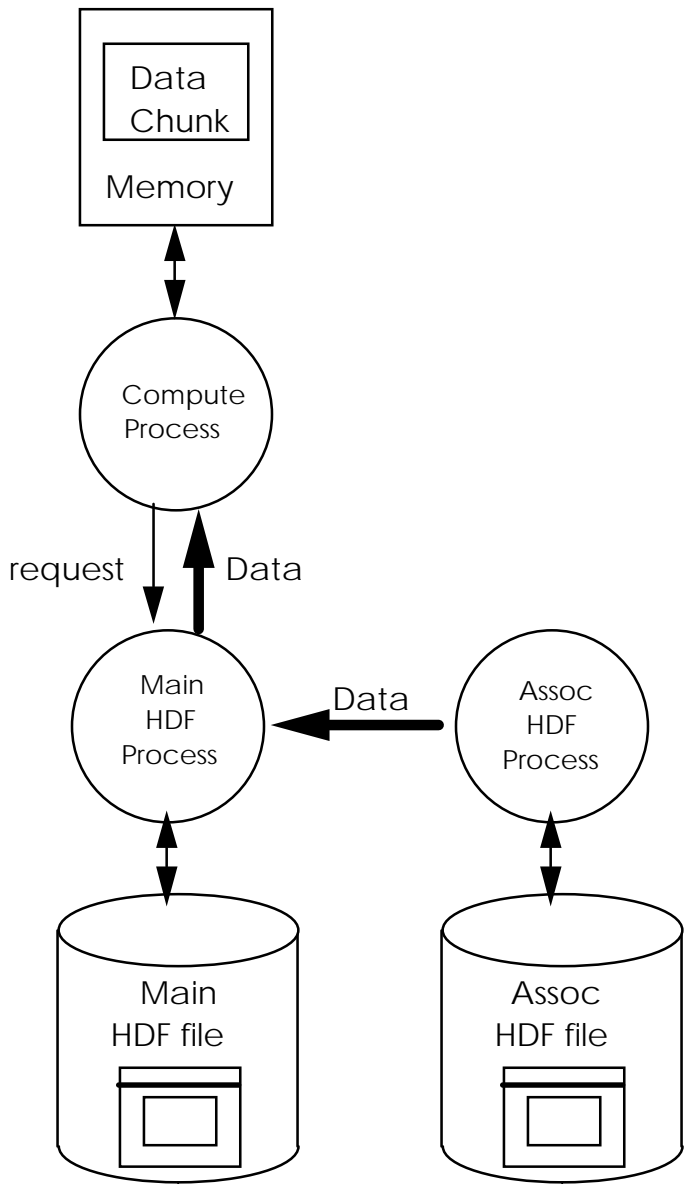


Figure 5-1 Central Hub Model

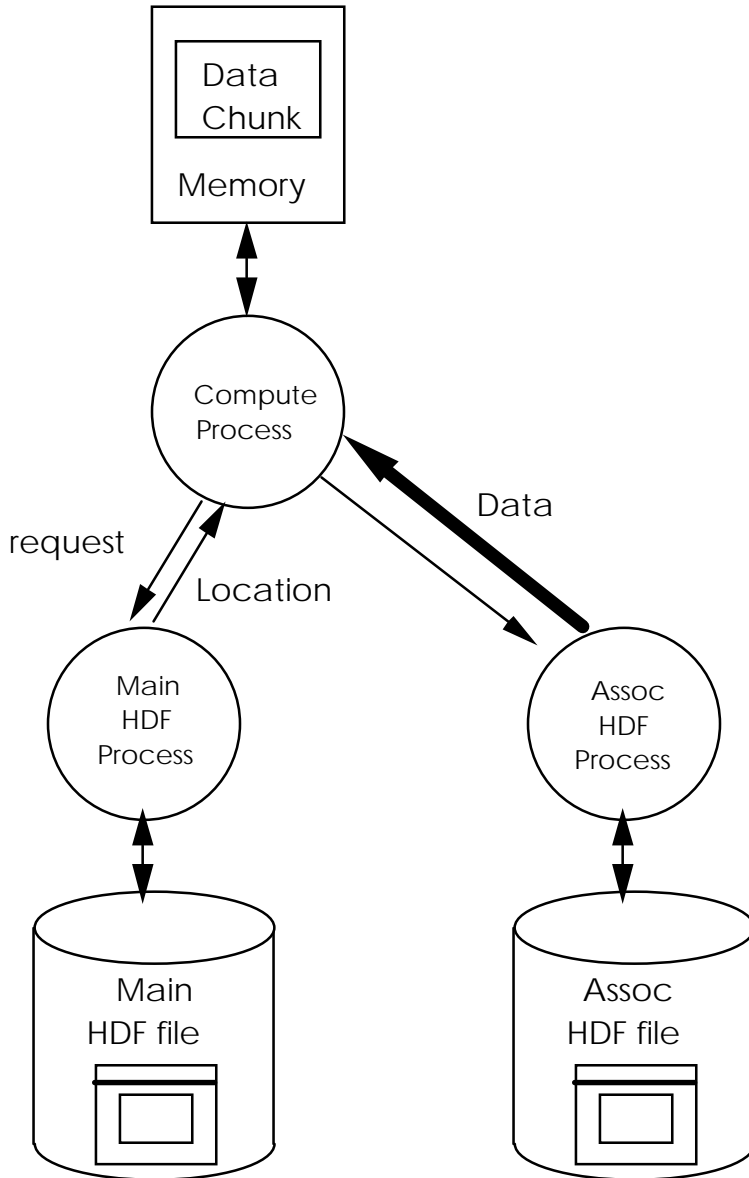


Figure 5-2 Switch Board Model

6. Conclusion

The *Multiple HDF IO* model is the most flexible one among the three models of distributed HDF files presented. The *Switch Board* model for I/O processing is a better approach than the *Central Hub* model.

We plan to design and implement the HDF-NOW base on these two models.

We plan to first implement the HDF-NOW on the workstations of the HDF group. They consists of SGI's, Sun's and HP's. We plan to build on top of the MPI message passing environment.