A Strawman Proposal

for the

Cluster Project

José Rogado

OSF Research Institute
2, Ave. de Vignate
38610 Gières
France
Tel: (+33) 76 63 48 66
E-mail: jose@gr.osf.org

Abstract:

This paper presents the rationale and goals for the cluster project, which aims at building a Distributed Operating System for managing networked workstations. The proposed foundation for this OS is OSF/1 MK-AD, the version of OSF/1 for multi-computers. This paper identifies the main areas where this system does not exactly fit the cluster environment, and goes further in determining possible solutions for extending it. The relationship between available user-mode clustering technology, such as DCE or PVM, and the proposed OS basis are also investigated.

1. Project Primary Goals

It is generally admitted that loosely connected pools of workstations present a very bad distribution of resource usage, for a certain number of reasons:

- bad allocation of computing power: while some workstations may be just prompting for commands, others may be completely overloaded.
- high percentage of idle CPU: workstations are mainly used in an interactive way, and human response time is known to be slow.
- overlapping and conflicting system administration: each workstation has its own system, its own needs for configuration, backup, updates, etc....
- bad reliability as a whole: one function executed by one workstation that fails has to stop even if others could continue the job.
- bad scalability: adding new workstations to the pool only benefits its

direct users, and not the group as a whole.

- no means for controlling parallelism: only advanced users can take some advantage of the inherent parallel structure of a workstation pool.

Extend the MC paradigm to Clusters

There is thus a need for managing networked pools of workstations in a much more optimal way: instead of being arbitrarily connected to a network, individual workstations can be considered as nodal resources of a networked multi-computer (**a Cluster**), and therefore submitted to global allocation and scheduling policies. The Operating System managing the cluster presents to users and applications the abstraction of a **Single System Image.**

The cluster project primary goal is then to extend the multi-computer paradigm to clusters of high performance workstations, connected by a high speed, low latency network. For a given pool of interconnected computers, this technology should provide:

- greater global throughput,
- easier system administration,
- better reliability,
- improved scalability,
- better usage of parallelism.

Use A Common OS Basis

Extending the multi-computer programming paradigm to a cluster is possible because, from an architectural point of view, multi-computers and clusters have many points in common. This has two implications:

- applications suited for a multi-computer can run (with some expected performance degradation), on a cluster, provided that both offer the same API;
- it is likely that many multi-computer OS components may, to some extent, be reused by a cluster.

It is thus also a goal of the cluster project to use OSF/1 MK-AD, the version of OSF/1 for multi-computers, as the foundation for building the cluster Operating System.

This system[1] is based on the NORMA Mach 3.0 µkernel, which runs on every node and offers a single logical µkernel interface across the whole multicomputer. Higher level operating system services are provided by a set of OSF/1 based user mode servers, running only on those nodes that need system services. All inter-node communication is done using NORMA Mach IPC.

From the point of view of users and applications, this system presents the following characteristics:

- it offers the complete standards compliant OSF/1 API;
- it provides a single system image;
- it is scalable to a large number of nodes.

---

1. We only give a very brief description of OSF/1 MK-AD here. Nevertheless, a good knowledge of this system will be needed for the rest of the paper. Please refer to a document called “The Architecture of OSF1/AD”, edited by the Research Institute.

2. Clusters vs. Multi-computers

Multi-computers and clusters present similar architectures: both are distributed, with no physically shared memory. But when we consider the real nature of its components, this resemblance disappears. In fact, cluster nodes are real computers, not special purpose hardware, and they are connected by a network, not a high-speed interconnection mesh. In particular:

- even using fiber optics technology, the network has lower bandwidth and higher latency;
- nodes may fail, since they are individual computers;
- the configuration of the cluster may change dynamically, since workstations can be added or removed from it;
- the network is an insecure medium, since the nature and identity of its users are not easy to establish;
- a cluster may be heterogeneous, since it may consist of different types of hardware.

Therefore we expect that a multi-computer OS cannot manage a cluster in a performant way, and that a certain number of extensions or restrictions are needed.

2.1 Cluster Definition

Before considering the extensions needed to adapt OSF/1 AD to clusters, we need to give a more exact definition of a cluster. According to its degree of homogeneity, integration and connectivity, we can consider at least the following fundamental types of clusters:

1) All nodes have the same CPU type, run the same kernel, and all the node resources are allocated to the cluster; in addition, nodes are connected via a private (physically secure) network. There may also be several connections to the public network, but the cluster is seen as a single machine. We will call it an integrated cluster.

2) All nodes have the same CPU type, run the same kernel, but nodes can also be used as normal workstations. Therefore nodes can also have individual connections to the public network. This is generally called a dual-purpose (or Schizo[1]) cluster.

3) Either of the above, but nodes can have different CPU types: it will be referred to as a heterogeneous (integrated or dual-purpose) cluster.

4) We can also consider highly heterogeneous clusters, where machines of different CPU types and running different Operating Systems can be connected, but this type of clustering is outside the scope of this project.

Obviously, the number of extensions needed for AD will not be the same for each type of clustering we'll be aiming at. At least in the preliminary phase of this project, the RI in Grenoble will be considering only the integrated cluster. The dual-purpose cluster will be University of Utah's contribution to the project, and HP Labs will focus on the heterogeneous aspects, as well as on high-speed networking.

---

1. As defined by Univ. of Utah.

2.2 Cluster Functioning Modes

In the following sections we will consider the extensions needed for OSF/1 MK-AD to manage the integrated cluster only. The nature of these changes also depend on the mode that we expect the cluster to function, namely:

- multi-computer mode: cluster completely simulates a multicomputer in its static configuration, failure modes, security, etc...
- single point of failure mode: the cluster is dynamically reconfigurable, which means that slave nodes can be added or removed from the cluster, but the failure of the root file server or the process master make the cluster stop.
- no single point of failure mode: the cluster can have any of its nodes fail without halting, although it may suffer performance and functional degradation.

OSF/1 MK-AD already supports the first mode, since Mach NORMA IPC has been extended to handle simple network protocols. This mode is actually used in multi-computer simulation for debug purposes. The following two modes though will require substantial extensions in both µkernel and user mode servers.

3. Micro-kernel Extensions for Clusters

NORMA Mach 3.0[1] functionality distributes the µkernel interface across node boundaries, giving the abstraction of a single unified µkernel across the whole cluster. This mechanism however was designed for the multicomputer case, which means that it does not take into account the networked character of the cluster. The following aspects need to be addressed.

3.1 Dynamic Configuration

NORMA does not support dynamic configuration: the cluster topology is acquired at boot time by each kernel from a server running on a non-cluster machine, and subsequent modifications need a global shutdown and reboot. We believe that dynamic configuration can be achieved with a reasonable amount of work. Adding nodes doesn't involve any cleaning up, so this should be the easiest part. Removing nodes, provided that it is done in an announced way, involves more work, which perhaps can be done without changing the current design of Mach NORMA IPC and XMM. Anyway, here is an initial list of problems that need to be solved:

- Information about cluster the configuration needs to be obtained dynamically, perhaps through a special NORMA Mach call: *norma_host_configure* with add and remove options; this call should be executed at bootstrap, after reading a configuration file, and a protocol should be followed to decide on which nodes being on the list should be considered as up and running.
- In case of node shutdown, the set of remaining kernels need to be informed so they can correctly discard all references to resources on this particular node.

---

1. The only available written (and out of date) description of the NORMA Mach 3.0 functionality can be found in Joe Barrera's paper in the 2nd Mach USENIX Workshop Proceedings (Monterey, Nov. 1991).

- Remove all origin-site dependencies on port migration, which perhaps conflicts with global lazy-evaluation techniques used.

A simple solution for adding new nodes to an already running cluster was implemented, with minor µkernel changes. This is required to allow any external machine to be used as debug node for the cluster system.

3.2 Improving Network IPC Protocols

The protocol currently used by NORMA remote network IPC, a scaled down version of UDP/IP, was designed for simulation purposes and is not adapted to intensive usage. According to the opinion of the x-kernel team from the University of Arizona[1], a global rework of the Mach IPC mechanisms and interfaces seems to be needed to improve the network protocol performances. A preliminary evaluation of the work needed to port this protocol framework to the Mach kernel has been done in Cambridge[2]. This work is likely to be pursued in Grenoble with the evaluation of other possibilities.

An in-kernel version of the x-kernel framework will probably be available from University of Arizona earlier next year. In the meantime, improvements to the current protocol used by NORMA Mach IPC are underway, which include retransmission, windowing, and message clustering.

3.3 Node Failure Support

The current NORMA µkernel does not support node failure. In case of node crash, all the cluster needs to reboot: node failure is not clearly detected, and the current IPC implementation does not allow easy port right disposal. This last problem is due to the fact that Mach IPC connection model implies keeping state: each IPC end holds port rights. Information about who holds port rights is kept in a distributed non-redundant fashion: a node that crashes takes with it information that is not available anywhere else.

Thus, node failure support at the µkernel level implies the following:

- a protocol for node failure detection and advertising,
- redesigning the way that port right information is kept.

On the other hand, the current version of the µkernel doesn't offer mechanisms for building fault tolerant servers. Among others, the possibility of sending messages to multiple ports (multicast), seems to be an important request to implement fault tolerance through replication. According to Franco Travostino, an evaluation of the possibility of using part of the ISIS protocols to address these problems should be done.

3.4 Security

The secure nature of Mach IPC is broken by the fact that it uses an insecure network medium, since an intruder can access the network, and pretend to be someone having send rights on privileged ports.

In order to restore security, additional non trivial extensions are needed. But since we are only considering the integrated cluster, where Mach IPC uses a private, physically secure network, we

1. See Sean O'Malley's presentation at the first cluster meeting held in Cambridge in July 92.
2. An internal RI document has been written on that topic by Franco Travostino.

won't treat that issue for now.

3.5 Cluster Micro-kernel

In conclusion, extending the µkernel for cluster support is a whole project per se. Some part of this work, like supporting dynamic configuration and integrating the x-kernel framework in the Mach µkernel, could be done in Grenoble. This would correspond to the definition of a first step in Grenoble's contribution, which would be to provide the distributed µkernel for integrated cluster applications. Although this kernel wouldn't offer fault tolerance, it would have the correct basic design to implement it.

4. Cluster Process Management

Several distributed process management implementations exist and are well described in the OS literature. It is not the scope of this document to go through any of them, but since process management is still not part of OSF/1 MK-AD, its worth expressing some expectations on this area.

4.1 Basic Mechanisms

This is perhaps one of the most important functions needed in a cluster OS. In fact all the considerations about increasing throughput by exploiting idle cycles are based on a global load distributing strategy for process scheduling. In order to implement it, cluster process management must at least provide the following mechanisms[1]:

1) remote execution of processes across nodes;

2) process migration (suspend on one node and resume on another);

3) process synchronization and signaling between nodes.

4) global identifying of processes across the whole cluster (global naming);

5) information about the load on each node, to be used by global load balancing and leveling algorithms.

4.2 Architectural Requirements

On the other hand, the process management service has to follow some architectural constraints in order to fulfill performance needs and provide a certain degree of fault tolerance (such as single node failure):

1) it is based on a distributed (peer-to-peer) collaboration of process management servers or agents running on nodes rather than in a centralized one;

2) it uses NORMA µkernel abstractions to identify resources attached to processes, so that migration, garbage collection, naming and fault tolerance can be done using available mechanisms.

3) it is able to accommodate node failure and still provide some coherent subset of the global process state and management (does not rely on

1. As stated before, we're only considering the integrated cluster functionality.

central information),

4) it is able to restore the totality of its functions when a node comes back to service, without global system reinitialization.

On top of these mechanisms, the global resource allocation services provide load balancing and levelling across the whole cluster. For now, we'll just concentrate on mechanisms and not on policies, with the constraint that the available mechanisms should be general enough to allow easy implementation of different global policies.

4.3 AD Process Management

As stated previously, the native OSF/1 MK-AD system doesn't currently provide this kind of model for process management. Processes are all tied to one central process manager, running on a master node, and all its process related calls are handled by this server. Remote execution is only done through the use of *norma_task_create*, which allows creation of tasks on remote nodes.

However, the AD system can be enhanced with third party software, which provides real distributed process management facilities. This software is available from Locus, and is called Transparent Network Computing (TNC).

4.4 Locus TNC Process Management

We don't have a very detailed description of TNC features, nor can we access its sources for evaluation. We can only give here a brief overview of its architecture, mostly based on Locus presentations and conversations we had with some of its developers.

1) Locus TNC process management is based on a collection of logically equivalent servers running on every node.

2) They maintain a real distributed peer-to-peer interaction to provide a global process naming space.

3) Processes running on a node always contact their local process server for their process management system calls.

4) TNC is implemented using NORMA functionality, such as inter-node IPC, XMM and global port identifiers.

5) TNC is inserted into AD by the use of the vproc layer, which is a clean well defined interface similar to the vnode layer.

6) TNC supports partial node failures, provided that the µkernel underneath does.

From the functional point of view, Locus TNC provides most of the mechanisms described above as being essential to cluster process management:

1) remote execution of processes;

2) process migration;

3) remote process synchronization and signaling.

4) global process naming.

It is also worth mentioning that originally, the Locus technology was designed for clusters, and that it was adapted to AD in the framework of the multicomputer project.

We lack information on the global policies that TNC builds on top of these mechanisms. So, although this technology seems promising, a deeper evaluation should be done, mainly in the cluster environment. So, it is part of the cluster project intermediate goals to find a workable solution between Locus and OSF RI, where Locus technology for process management, already ported to AD, can be made available. We suggest that this evaluation should form part of the Grenoble RI contribution to the cluster project.

4.5 Other Techniques

This should not preclude us from examining other process management technologies. For instance, SPRITE is also known to provide a similar type of fuctionality. Recently, there has been an implementation of a SPRITE server on the Mach µkernel environment. However, this server has not reached a stable state where it is usable. The latest version does not support process migration. Some incompatibilities were found between the fact that Mach external memory managers cannot be aware of all the memory created by processes (mainly temporary and shadow objects) and the need for the Sprite server to be the only memory manager for all its processes memory. The person that did the implementation, Mike Kupfer, is no longer available for further work on this subject since he changed jobs recently. According to him, porting parts of the Sprite server to the OSF/1 environment would require a large amount of work. However, the load balancing daemons seem more likely to be portable.

5. Cluster File System

In this section we will discuss the Cluster File System. We will start by describing an existing base, the AD File System and we will then examine the extensions it needs in order to fulfill the cluster needs. This will lead us to a more general model of what could be envisioned as the ideal cluster file system. On the next section, a more realistic solution is proposed, which is based on existing OSF technology.

5.1 The AD File System

The OSF/1 MK-AD File System (ADFS) was specially designed for multicomputer usage. It offers a single file name space across the cluster, based on a collection of file servers running on nodes providing file storage. Servers are assembled by remote mount points, which are similar to OSF/1 mounts, where remote servers are named using NORMA global ports. A similar mechanism is used to access remote devices. Although this design seems well suited to distribution, its implementation is not well adapted to the cluster environment. Mainly:

1) The collaboration between servers is based on a master slave model: the root file server is the master, and all the servers managing the rest of the file space are slaves; therefore, if the master server fails, the other servers cannot functionally survive.

2) Remote data caching is done by using mapped files implemented partially in the emulator; however, there is no provision for caching file attributes, which means that every operation involving attributes has to go across the network if the server is remote.

3) Message path optimization is done by having the emulator directly store ports to opened files; this architecture constraint implies that all the

caching and pathname lookup optimization have to be done in the emulator (see below).

4) There is no provision for replication: since ADFS doesn't use the vnode layer, it cannot benefit from possible OSF/1 extensions like stackable file systems, which seem to be a good approach to implement replication at the file system level.

It is thus expected that important extensions or modifications need to be made to support cluster functionality, and therefore we consider that the AD File System is not directly usable in the cluster project. Next paragraph will focus on how ADFS architecture should be reorganized in order to fulfill the cluster needs.

5.2 Preliminary Architectural Requirements

The current ADFS implementation is strongly biased by the existence of an emulator. Important parts of its functionality, like mapped files, translating open files to Mach ports, and system call interruption are partially implemented in the emulator, or rely strongly on it.

The emulator problem

In our opinion[1], the emulator introduces an extra level of complexity in the architecture and resultant code, mainly because state has to be shared amongst servers and emulators, on an already distributed complex environment. We would like to avoid that in the cluster project. Therefore, we consider that the no-emulator work being carried out at Grenoble is of direct relevance to the cluster file system.

The current ADFS functionality has been designed to allow direct interaction between applications and file servers, possibly located in other nodes. This interaction is based completely on the emulator existence, which acts as a proxy for the file system server in the application address space. If the emulator is removed, all contact with remote file servers has to be mediated by a local agent, which will be responsible for maintaining information about files. This will result in a complete re-architecture of the AD file system.

The benefits we expect to obtain from this approach are the following:

1) All the knowledge and state about files on a node is centralized in a trusted and independent entity, that can be a server or agent depending on implementation considerations[2].

2) File system optimizations, like mapped files, can be implemented in a simpler way inside the server, and benefit from global solutions.

3) The AD file system can be inserted in the vnode layer architecture and therefore benefit from global existing caching techniques and optimizations (like directory lookup caches or prefix tables).

4) Future developments of the OSF/1 integrated kernel (namely stackable and log based file systems) can be directly used to add replication and checkpointing techniques to ADFS.

---

1. See also Simon Patience's presentation "Life Without an Emulator" at the Munich Members Meeting in April 92.

2. In the following we will only refer to this entity as the local file server, although for performance reasons, it might as well be an agent implemented in the kernel address space.

On the other hand, keeping information about files in the local server may need more work when a process with open files migrates. In that case, files opened by a migrating process need to be reopened by the server running on the new site, so that the underlying structures for holding state are created. Besides, locally opened files need to be associated with ports, so that subsequent access by migrated processes is possible. Another small advantage of having the information about files in the server is that this association can be done lazily, when a process holding references to local files migrates.

The vnode layer

As we have just seen, one immediate and natural consequence of removing the emulator will be to insert ADFS under the vnode layer. In fact, the server part of ADFS is implemented somehow above and below the vnode layer: actually every file system type has to know about it. The reason for this is that the vnode layer does pathname lookup one component at a time. ADFS, on the contrary, forwards the system call in progress with the remaining pathname as a whole to the file server referenced by the "mounted on" vnode.

This design is well adapted to the fact that file information is kept in the emulator: no state remains in a server traversed during a pathname translation if the target file lives in a different node. However, in the absence of emulator all the file activity on a given node is handled by the local file server, therefore state about files has to be stored locally. In our opinion, the vnode abstraction provides a well understood and proven model for encapsulating file dependent information, and thus, file related information (like the server port, attributes, etc...) should be kept on the file system dependent part of the vnode.

Although we think this is the right way to go, there are still a few problems that need further investigation before we are able to propose a complete ADFS redesign:

1) Pathname lookup: the current vnode interface only allows one component at a time lookup, which increases the number of messages for pathname resolution.

2) System calls will be forwarded at the vnode level only, which somehow decreases the interaction semantics between servers.

3) Servers will have to export interfaces at two different levels: the system call interface and the vnode operations interface; this would increase its complexity, since there would be little code in common between these two interfaces.

4) Data caching: if we follow the vnode model for caching data, we end up using the traditional buffer cache; this can possibly be avoided by allowing mapped file access to remote files and relying on NORMA XMM functionality for data coherency.

5) the credential service has be redesigned to handle vnode level identification.

Some of these problems may be hard to solve, but we think this re-architecture is needed in order to simplify and enhance the current file system. Perhaps a hybrid architecture could be envisioned, where ADFS would still use part of its original functionality for pathname lookup and system call forwarding, yet would also fulfill the "file system type" semantics imposed by the VFS interface, and would therefore benefit from global solutions for file access and caching optimizations.

5.3 Distributed Server Cooperative Model

In this section we will address the global model used for achieving single file system image in the cluster environment. This model has to follow certain rules in order to bring the expected benefits, which are mainly increased throughput and a certain degree of fault tolerance. The first step towards this goal is to base the file system architecture on a distributed and independent set of cooperating servers.

One File Server per node

A File Server should run on each node having disk resources, and should manage its own file system. For performance reasons each node should have a local cached version of the most frequently accessed directories of the root. Coherency should be maintained by a distributed algorithm. But since the root file system (except /tmp, see below) is not subject to change often, this algorithm can be relatively simple and optimistic.

The /tmp directory

The way the /tmp directory should be treated is a subject of investigation: we believe that for performance reasons there should not exist one centralized version of it. On the other hand, having one /tmp directory per node and keeping it coherent is expensive, because it changes a lot. But since only temporary information is supposed to be there, we can admit that only programs running on different nodes and communicating through files in /tmp need coherency to be observed. So a solution between having one independent /tmp directory per server and a complete coherency scheme has to be found. One possibility would consist in having the concept of current working directory extended to the /tmp directory. Families of processes, which are the ones more likely to share temporary files, would inherit a common handle for /tmp. Therefore pathname resolution for that directory would end up in the same place, even if the processes are running on different nodes. A way for changing the handle for /tmp has also to be provided. Special cases like window systems and its clients (actually the X server) use special files (fifos) in the /tmp directory to establish meeting points. A special treatment would be needed for these cases, that could possibly be based on the fact that the files are special and therefore their existence should be propagated to all /tmp servers. In any case, this is an open issue.

Building the global view

The static OSF/1 mount model used in ADFS is not well adapted to a dynamic reconfigurable file tree. If we follow that model, each time a new node comes up, a new mount action has to be performed in each node so that the new exported file system takes its place in the hierarchy. A dynamic mount action, based on information provided by the joining node should be envisioned to make the file tree change accordingly. If the emerging file system overlaps with already running ones, then it should be considered a replica for it. Otherwise the tree should be extended and the mount points should be created accordingly. A coherency validation protocol should be followed to guarantee that the current view is still up to date. Besides, when a node goes down, if it is the only manager of that part of the file system, the corresponding subtree should disappear from the global one to avoid hang ups and time-outs on non existing files.

Locus TNC has a feature called the cluster mount service that implements this model. We don't have enough information about it, but it would be interesting to evaluate its functionalities.

File Replication

Replication of servers managing critical parts of the file system is the only way to support node failure. We believe that the implementation of replication at the file system level should rely heavily on Mach mechanisms. For instance, representing a file by a port and sending a dead_name_notification to all the holders of send rights to this port when its node goes down would help to keep the file system coherent without needing to probe servers. On the other hand, at the file system server level, replication can be easily implemented using stackable vnodes. Each level on the stack points to a different replica of the file. Combining this with Mach ports notifications allows for a flexible design where replicas can be dynamically created and removed in the file system space.

Using existing technology

Implementing this model from scratch in the OSF/1 AD system is not possible in the framework of this project. A large amount of manpower would be needed, that is not currently assigned to the cluster project. Besides, many aspects we ignored at that level of description cannot even be evaluated. On the other hand, we know that several distributed file systems examples are built using this model or a very similar one.

Among them, the Locus Distributed File System is perhaps the one that presents the closest approach, at least from what we can read in the available literature[1]. Perhaps an evaluation of the work needed to port that file system to AD should be done (by LOCUS, OSF?), even if the resulting prototype would not be part of OSF/1 MK-AD. This could be, as in the multicomputer project, some added value that clients of the cluster OS would get separately from LOCUS. OSF/1 AD would only contain a minimal set of hooks allowing a fast integration of the distributed file system functionality.

Another approach to the problem would be to use an off-the shelf OSF product available without restrictions: the DCE/DFS file system. We will consider that solution in the next section.

5.4 RI Contribution to the Cluster File System

The RI contribution to the cluster file system work should consist in applying to the OSF/1 MK-AD system the architectural changes we described: removing the emulator, finding a correct solution to implement ADFS so that it is inserted under the vnode layer, and add stackable vnodes. This last feature will probably be obtained from the OSF/1 system main line, since there are plans to include it in OSF/1.2[2].

The resulting advanced development prototype would be a basis on which other parties could implement higher layers of functionality, like replication, fault tolerance, single global view, etc...

Another possible solution is to start with an OSF product that partly fits the cluster file system functionality (the Distributed File System - DFS) and adapt it to the cluster needs. We will consider this in the next section.

---

1. “The LOCUS Distributed System Architecture” by G. Popek and B. Walker.
2. George Feinberg from OSF Engineering is planning to include stackable vnodes in the next release of OSF/1.

6. The DCE/DFS model

DFS is the Distributed File System provided by OSF as part of its Distributed Computing Environment offer. It is derived from Transarc AFS, technology which is widely used in the scientific computing community. Although the client-server model DFS implements is different from the one we are envisioning for the Cluster File System, it contains an interesting set of functionalities that could be used as a basis for building it.

6.1 DFS Architecture

DFS is based on a client-server paradigm: a set of machines, the servers, provide a distributed repository for files, to which another set of machines, the clients, access via secure standard protocols. Servers are configured into a logical administration unit called a cell. Clients may or not belong to the same cell.

Client

The client side is implemented as another file system type under the VFS layer, which had to be specially modified for it, and some DFS specific system calls added to the host operating system. It consists mainly of a cache manager, which uses the local file system for caching files (calls to the local file system go through the VFS layer). Cache coherency is managed between the cache manager and the servers through the use of tokens, with multiple reader single writer logic.

Communication between the clients and servers is done via authenticated in-kernel[1] RPC, also a DCE component, which enforces security. Access to files is governed by the use of Access Control Lists.

Server

The server side is implemented by extending significantly the host OS. It consists mainly of the file exporter which services RPC requests, contains the token manager and accesses the server's file systems through the VFS layer. Local storage is provided by a new type of file system called the Log File System (LFS), also integrated under the modified VFS layer. LFS is a transaction based file system, which is always in a coherent state. Most of the DFS functionality is based on the fact that it uses LFS as the basic storage for files, since LFS supports ACLs and the notions of several filesets (file subtrees) per aggregate (partition) which are used in replication.

Other (user mode) servers are also needed to implement the DFS server functionality:

1) the Fileset Server, which manages filesets;

2) the File Set Location Data Base (FSLDB), containing information about file set location;

3) the Replication Server which manages replication.

A server host can also be configured as client and access its own resources through DFS.

Building the global view

The DFS file space is a hierarchical tree that follows naming conventions provided by another DCE component called the Directory Service. Each cell is visible under a global root and is named */../cell.name*. Access to the contents of the cell are done through the name /../cell.name/fs.

---

1. In-kernel here means it is implemented inside the host OS.

The configuration of a cell file system is done entirely by the servers: filesets are made visible in the exported directory tree by the use of persistent mount points, that survive server shutdown and crash. When a pathname translation traverses a mount point, the FSLDB is contacted and the location of the server for the mounted fileset is returned. Further lookup operations are performed one component at a time by the server containing the mounted fileset. In addition, a fileset can be replicated in several servers, which increases server availability for performance and provides for a certain degree of fault tolerance.

Explicit access to the DFS directory (or usage of symbolic links) has to be done by the clients in order to view the distributed file space. Once accessing a particular cell, clients have the view that the cell exports, which is configured by administrative means.

6.2 DFS Limitations

Having described the DFS functionality, we will now consider how it can be used to implement the cluster file system, and highlight the areas where the proposed model cannot be achieved.

The cluster file system model with DFS

If we assimilate a cluster to a DFS cell, the cooperative file system image described in the previous section cannot be achieved, since DFS treats nodes asymmetrically. Only server filesets participate in the creation of the global view, and client local file systems are not part of the file tree.

However, if all nodes can be configured symmetrically as clients and servers, with all local files exported to the cluster file tree as DFS filesets, a reasonable approach can be done. Currently, we don't have enough information about DFS to know if it admits a very large number of servers inside the same cell. It may scale poorly due to administrative problems or network traffic intensity. Perhaps a scaled down version of the server suite may be used, mainly if using the server code in many nodes dramatically increases licensing fees[1]. Some evaluation work needs to be done in this area, involving perhaps some discussion with Transarc.

Another solution is to configure clients with a local reduced bootstrap file system, not to be exported, and use all the rest of their disk space as a DFS cache. This solution applies to regular user workstations, that have small amounts of disk space, but cannot be extended to all client machines in the cluster, mainly because it does not allow cluster resource extension through the addition of new nodes. Workstations possessing large amounts of disk space should be configured as clients and servers. So we think that a combination of both solutions has to be used to achieve a correct approach to the cluster single file system image.

Dynamic configuration

DFS does not allow dynamic configuration. Mount points are persistent and don't change when a new server joins or leaves the cluster without explicit administrative actions (and perhaps server shutdown): new servers joining the cluster need to be explicitly mounted to be visible in the file tree. This supposes that the file tree configuration is static, which greatly reduces the cluster file system scalability.

Thus dynamic cluster configuration cannot be achieved without modifying current DFS functionality. A mechanism for announcing that an unknown node is joining the cell, or that some mounted one is leaving it, needs to be designed. All FSLDB's will have to be informed of the

1. Using the server part requires a license fee of several hundred dollars.

identity and location of the joining or leaving node. Some new administration tool will have to be invoked to create or remove the corresponding mount points, which will depend on the relative place the corresponding filesets occupy in the cluster file tree.

Handling frequently accessed files

As we mentioned in the previous section, some frequently accessed directories in the root file system, which change rarely, should be made local. DFS provides mechanisms for this, either by accessing them through a local server or by using the local file system as a cache. If the file system doesn't change, the tokens for its files won't be revoked, and the accesses will always be local, unless cache space is exhausted.

The */tmp* directory is a different problem: it is very frequently modified by file creation and deletion. However, as we've seen in the previous section, coherent global view of its content is not always needed. The current DFS functionality has no way to handle this problem correctly: if this directory is cached locally, the token for it needs to be revalidated every time another node modifies it. So a solution like the one already proposed has to be added to DFS in order to handle this problem correctly.

Access to remote devices

DFS provides no way to access remote special files. On the current AD File System remote devices are accessed by Mach ports, and once opened, direct references to the µkernel containing the device driver for the device are possible. A similar mechanism has to be inserted in DFS. This has to be evaluated.

Support for process migration

In ADFS, the fact that files are represented by ports allows for simple and efficient access to opened files by migrating processes. Send rights for ports representing files simply have to be sent in a NORMA Mach message to the task containing the process in the new node. The two intervenient µkernels will cooperate to migrate the port rights correctly. Such a mechanism is not part of DFS, and we are not sure that the internal DFS client representation of a file will have any meaning if we transfer it to another client. This issue has to be investigated and solved in agreement with the Process Management policy adopted for the cluster OS.

DFS use of RPC

In the reference versions of DFS, communication between clients and servers is done via an in-kernel RPC, which has to be inserted in the host operating system by linking DFS code against special libraries. This RPC, which is also a part of the DCE offer, uses NIDL as a stub generator and may be layered on top of UDP, TCP/IP or other equivalent protocols. It uses the DFS authentication scheme to ensure security, DCE naming for binding, and DCE threads for multiplexing purposes.

Although this RPC may be useful to connect heterogeneous Operating Systems, it introduces a large amount of overhead if we compare it with an RPC based on Mach IPC. Recent analysis of the DCE RPC indicates that it takes about 60,000 instructions to execute a 32 byte RPC between user mode client and servers[1]. This is to be compared with several thousands (~5,000) of instructions only, in the case of an RPC based on non optimized NORMA IPC. In the case of an inte-

1. According to Greg Buzzard from HP Research Labs, on HP/UX it takes about 66,000 instructions for the first RPC and 56,000 for each subsequent one using UDP, and 148,000 /57,000 using TCP.

grated or dual purpose cluster, there is no need for such an overhead, if we admit that the underlying basis is NORMA Mach 3.0.

There is thus a need to tailor the DFS RPC to the reality of the µkernel environment, and replace the transport mechanism it uses by the one on which we intend to base the cluster OS. Several approaches[1] can be followed to do this replacement, either by using Mach IPC as another transport mechanism, accessed through the same NIDL stub generator interface as others (TCP, ...), or replace the entire DFS RPC layer, NIDL stub generator, etc... by one built on Mach and MiG. Another approach would be to extend NIDL with knowledge about Mach ports and message formats, and simply link the DFS code with the new version of this DCE run time library. In any case, conversion between DCE naming and binding conventions and the ones used by Mach needs to be done.

In any case, integrating DCE RPC and NORMA IPC is certainly not a trivial task, and requires additional investigation and design discussion.

DFS is not part of OSF1/MK-AD

DFS is not part of the OSF1/AD-MK system, mainly because there is no stable version of this file system yet, and OSF/1 MK-AD hasn't yet integrated even its predecessor, the Andrew File System. So before doing any further work in this area, the standard DFS functionality has to be ported to the AD system. And Since DFS uses most of DCE functionality and components, porting DFS means porting the whole DCE suite, which is not a simple task.

To achieve this, we can count with the collaboration of the Helsinky University of Technology (HUT), which has started to port DCE on the OSF/1 MK system[2]. Once this system is available, porting DFS to AD (in the standard way) will be greatly simplified. After that, it will be necessary to achieve a closer integration of NORMA Mach IPC and DFS, as we have just described.

6.3 DFS Advantages

If a close integration of the DFS and NORMA functions can be realized, the resulting system will benefit from all the available DFS technology, which is well adapted to the cluster OS.

Existence as an OSF product

First we have to consider the fact that DFS, unlike other proposed technologies for cluster file systems, is an OSF product that is likely to be widespread. This has the advantage that no special agreements need to be done with external suppliers to include it in the cluster OS.

On the other hand, from the marketing point of view, presenting to customers and members an unique distributed file system interface is a great advantage. This file system would have two implementations, one adapted to closely coupled and integrated environments, the cluster, and another for heterogeneous networks, which could be made compatible. This way, interoperability would be increased and the OSF offer for Distributed Environments would be coherent.

Hierarchical Inter-Cell Trust

In the previous section we identified a cluster as a DCE cell for building the single file system image. Although this may lead to a approximation to the model envisioned, it introduces a configuration problem from the security point of view: the relationship between the cluster and the other

---

1. See D. Black's memo on that topic, dated January 91.
2. There is also a port of DFS to OSF/1.1 for the i386 done by Gary Fernandez.

non cluster machines, still belonging to the same geographical DCE cell is not clear. The notion of hierarchical inter-cell trust, which will be in DCE future releases[1], allows this relationship to be established in a simple manner. In fact the cluster can be considered as a DFS sub-cell, which in turn would be part of the local cell, on which other DCE hosts will exist. Therefore the limits of the cluster will be logically defined by the sub-cell entity, and all security problems between cluster and non-cluster machines will be submitted to general DCE security and authentication policies.

Security

One of the most interesting aspects of DFS is that it allows secure interaction between client and server on intra or extra cell accesses. The security model is based on an authenticated RPC that uses unique user identifiers (UUID's) and Kerberos tickets for identifying RPC partners, and on the use or Access Control Lists for checking authorization on file operations. This security model has to be preserved when switching to another type of RPC, based on NORMA Mach IPC.

In an integrated cluster this is not an issue, since only intra-cell RPC will use NORMA, and the cluster is considered as a separate entity with a private network for intra-cell traffic. Security will only be enforced on those nodes acting as gateways to the external network (see next section on Cluster Networking).

However, on a dual-purpose cluster, where each node may have its own normal workstation personality, and participate in non cluster accesses to the network, careful separation of both worlds has to be achieved. This is part of the dual-purpose cluster project work.

File System Replication

DFS allows read only fileset replication. Although this may not be enough to implement fault tolerance, it is a first step towards it. If DFS replication can be layered on top of fault tolerant mechanisms offered by an enhanced NORMA µkernel, then higher levels of reliability may be achieved. Replication also has a positive impact on performance, since it allows load sharing.

Log File System

DFS uses a Log File System for storing server files. This file system is always in a coherent state, and log recovery allows higher data integrity in case of a crash. Besides, fileset cloning is supported as a new operation which is part of the extended VFS layer, which means that at any time a snapshot of the current version of a file set can be created. This is a something that can be directly exploited by global checkpointing policies.

Extended VFS layer

DFS introduces substantial extensions to the widely used VFS interface. These extensions include operations to manipulate aggregates (which are the equivalent of disk partitions), filesets, and general credentials based on Access Control Lists. On the other hand, it introduces an additional level of vnode operations, the wrapper functions, that handle token synchronization needed for distribute state coherency, and then call the traditional vnode operations. The fact that a layered approach has been followed to introduce new functionality is a good design decision that facilitates evolution and code understanding.

---

1. See Note on Hierarchical Trust Relationship by J. Pato from HP, which is an OSF DCE SIG Document.

6.4 Conclusion on DFS

In conclusion, we may say that DFS unchanged cannot be used as the cluster file system. But a solution consisting of integrating DFS with the underlying µkernel functionalities in order to build a well adapted cluster environment seems to be attractive, mainly if it may correctly use all the available DFS advantages just described. On the other hand, this solution, as compared with others consisting of using external vendor technology, will lead to a greater project scope and credibility.

To do so, the most important achievement will be to use the enhanced NORMA Mach IPC as the transport mechanism for the DFS RPC. Other advanced µkernel features we described as mandatory for the cluster environment, like node failure detection, correct port rights disposal, fault tolerance, etc... should also be used to adapt existing DFS functionalities to the cluster model.

Since DCE and DFS are currently being ported to OSF/1 MK server at Helsinky University of Technology (HUT), there is a good chance that a server based standard DCE/DFS will be available for evaluation in Grenoble within a reasonable amount of time.

The next step will consist in porting DFS to OSF/1 AD and integrate DFS with Mach IPC. To do so, we first need to implement some of the architectural changes described in Section 5.4: remove the emulator, extend the vnode layer, etc... Only after that could ADFS be replaced by NORMA based DFS (or made available optionally), while some of its mechanisms, like representing opened files and remote devices by Mach ports will certainly be retained.

Another faster alternative to validate this solution, will be to start with the no-emulator OSF/1 MK, merge the DFS port from HUT (which will be easier since the code basis is the same), and integrate DFS with Mach IPC. This has the advantage of avoiding interferences with AD architectural issues, but will lead to a intermediate version which will have to be merged with AD later. Some preliminary evaluation on which of those two solutions should be followed, or if both could be achieved in parallel, has to be done prior to any development activity.

7. Cluster Networking

The current OSF/1 MK-AD system provides no distributed solution for networking, although existing third party software (Locus TNC) can be added to it in order to achieve distribution of network functionality.

In this section we will describe the model that we envisage for the integrated cluster networking, and then try to evaluate some possible solutions for its implementation. The dual-purpose case will probably be addressed in a later version of this document.

Integrated Cluster Assumptions

As we stated in Section 2.1, nodes on an integrated cluster are connected through a private network, which will mainly be used to exchange NORMA Mach messages. All its external network activity will be done through specialized network nodes that will be dual ported, with one connection to the cluster network and another to the rest of the world. These nodes can also be considered as gateways.

Requirements for Cluster Networking

The requirements for cluster network software are the following:

- throughput: several instances of the available protocols must run concur-

rently on the network nodes.

- single image: all the instances of the protocols must act globally in order to offer a unique host image.
- node independence: processes with active network connections should not be bound to network nodes and should be able to migrate.

In order to implement this model, distributed collaboration between several system components has to be achieved.

Global System Architecture

We assume that every cluster node runs a µkernel and a local server or agent that may respond to system call requests from applications. In addition, this server may also have network protocol implementations on it, even if the node has no external network interface. In this case, the main components that are involved in network activity, and types of communication between them, are the following:

1) Network device drivers and packet filters: in our case these are part of the Mach µkernel handling the network devices, and therefore may be accessed by messages from any node through NORMA Mach IPC.

2) Protocol Handles (Sockets or Streams): these are located inside OS servers and are accessed through a procedural interface by the system call layer (outbound) or the top of the protocol stacks (inbound).

3) Protocol Stacks: these are accessed by the protocol handles by procedural interfaces (outbound), and by messages from µkernels handling network devices (inbound); access between contiguous layers of the protocol stacks is also done in a procedural way.

To achieve single network image with multiple network nodes, additional message based communication channels between system components and protocol layers need to be created for sending data, synchronizing and managing global table coherency. Several approaches can be followed, depending on the layers we choose to modify.

Unmodified Protocols

In this case, the protocol stacks will run unmodified on servers that will be located on network nodes. Client nodes will create local dummy protocol handles that will be connected through a message based channel to real protocol handles on the server side, after a given network server is selected (based on fixed partition routing, round robin, load, ???). The interface between the local dummy handle layer and the remote protocol handle implementation will be done by NORMA Mach IPC. Protocol handles will be associated with Mach ports on both sides of the channel, since both sides have to implement client and server interfaces (asynchronous requests can be done on both directions). Network accesses done from applications running on network nodes follow the usual procedural interface.

In case of process migration, only the system call side of the channel has to do port migration and re-creation of structures to insert the references to the remote handle (equivalent to file reopening).

This architecture, however, binds the handle to the node where the particular instance of the protocol it gives access to is running. No other network traffic for this particular handle may exist on the cluster, which may be a severe limitation. On the other hand, foreign hosts having done

address resolution will associate this physical interface with the logical cluster address. To avoid this restriction, additional communication between protocol servers for keeping coherency between address tables has to be implemented, which occasionally will introduce changes to the existing protocol code.

Modified Protocols

Additional investigation has to be done in this area before more concrete architectural propositions are made. In the meantime we can refer to a document by David Black (Internet Sockets, October 91) describing possible Internet protocol changes for the Multicomputer case, which is similar to the integrated cluster, if we don't consider node failure for the moment. Two solutions are proposed.

The set of sockets

This solution allows sockets not to be bound originally to any specific network interface: a server, which is on the node where the application runs, creates a local socket, and binds it to one of the existing interfaces at routing time (thus this local server needs to have routing information). At that time, the server managing the node with the chosen interface, is asked to create a local socket, which will be associated by a Mach channel to the client one. In cases where no explicit routing is done, one socket will have to be created on each available network server, which will be associated on a "socket set" (by Mach "glue"), from which the client side socket will be the master.

One way to implement this solution is to have one multicast address for the whole set of interfaces connected to the same physical network, and to have efficient packet filtering (i.e. based on source machine address). Packet filtering will be enabled on demand after a decision has been made about which interface manages which connections. This solution allows "listening" on TCP/UDP ports using every available interface, which allows parallelization of receive operations.

Switch at the top of IP

This solution creates a communication channel between the lowest level of the Internet protocols (IP) and its adjacent levels (TCP or UDP). In this case, a server containing a copy of the protocols runs on every node. Sockets are created on the server that is local to the application issuing the request, and only its copy of the protocol stack is active for one socket, for the levels above IP. All the incoming packets for this port coming from several copies of IP running on the network nodes would be sent to this server through Mach IPC. This solution also allows listening on multiple interfaces, since packet assembly and global TCP state is kept in only one server.

To implement this solution other coherency problems have to be addressed. All the existing network servers would need to collaborate to maintain coherency of global tables, like the ARP and routing tables. On the other hand, identifying endpoints needs some kind of nameserver to be implemented, that would allow the IP level, which is receiving packets for a given endpoint, to identify the server where the right socket lies: this service would have to go through all the servers in the cluster and all the sockets created on them, which may lead to considerable implementation and scaling problems.

Other solutions

A possible solution, that would allow easy creation of communication channels between layers of different copies of network protocols running on cluster nodes, would be to have a modular

and layered implementation of protocol services, like the one the x-kernel allows. This way, inserting Mach channels between different protocol intermediate phases would be straightforward, by using x-kernel micro-protocols assembled by an x-kernel virtual protocol for deciding whether the adjacent layers are local or remote. This solution however, implies the replacement of the current BSD network protocol implementation present in the OSF/1 based servers by one done in terms of x-kernel abstraction.

Cluster Network Address Resolution

From the external network point of view, the cluster should be considered as a single machine, thus will respond to an unique logical address (per local network address). Depending upon the chosen implementation for the cluster network software, the Address Resolution Protocol may map this logical address to several physical network address corresponding to the different network nodes, or to a single multicast address which will be recognized by all the nodes simultaneously. This solution may be implemented with the use of packet filtering present in the Mach µkernel.

Locus TNC solution

TNC implements cluster networking on the OSF/1 MK-AD basis. According to Locus, this facility offers the following functionalities:

- Sockets are allowed to migrate, although end points stay bound to nodes.
- Existing protocols and applications are unchanged.
- Communication servers can run on subset of total nodes.

We don't know exactly how Locus implements their cluster networking, nor do we know how efficiently does it take advantage of having multiple network interfaces. If some agreement is obtained between Locus and OSF, we suggest that some investigation is carried in this area.

8. Exploring Other Clustering Technologies

During the previous sections, we described the functional extensions needed to adapt a multicomputer Operating System to a cluster environment. In this section we will briefly consider other existing technologies that also tend to offer an integrated view of a pool of loosely coupled computers, although they don't go so far in achieving a single system image. Among many existing products and research prototypes, we selected those two, that for several reasons, we can't ignore in the cluster project: PVM and OSF's DCE. We describe below their important features and possible relationship with this project.

8.1 The Parallel Virtual Machine

The Parallel Virtual Machine (PVM) System is a tool for building and controlling distributed concurrent applications on heterogeneous networks of computers. It considers the underlying pool of available machines as a single entity with computing and storage resources, and provides primitives to control them that can be embedded in different programming languages. PVM is publicly available from the Oak Ridge National Laboratory.

Application programs view PVM as a general parallel computing interface that supports shared memory, message passing, synchronization, with several degrees of transparency[1]. In

1. For more detail, see paper "PVM: A Framework for Parallel Distributed Computing", by V.S. Sunderam.

addition, PVM provides a graphical front end, called HeNCE, to administer and configure hosts, and execute, monitor and debug applications. The logical processing unit for PVM is the *component*, which in most implementations maps to the concept of a user process.

The most interesting feature of PVM is its portability and easy configuration. For this reason, a large number of parallel applications are based on it, and should be able to continue to use it even if the underlying Operating System provides a higher degree of cluster integration.

There is thus a need for the cluster OS to support PVM. This support can be made optimal if we use cluster OS specific functionalities to implement the PVM interface. This way, existing unchanged applications would gain on performance when migrated to the new cluster environment.

PVM generic implementation

The PVM support software executes as a user process on each host of the declared pool (which can change dynamically). The support process is called *pvmd* and is responsible for all application components that execute on a host. There is no central pvmd process, since all instances have global knowledge of the PVM state.

PVM assumes only that unreliable, insecure and point-to-point transport is available. Reliability, sequencing and broadcasting are built-in PVM facilities. Therefore, almost all the versions of PVM use UDP sockets for inter-host communication. Binding is done using the concept of the "well known port" for initial inter component messages, which is then replaced by ports referencing registered (*enrolled*) components, provided by the PMV daemon. Local communications between PVM components may use several services, like AF_UNIX sockets, if they are available. Otherwise, the generic version uses loopback UDP facilities.

PVM implements shared memory by using a file on a distributed file system visible by all hosts. When a shared memory region is created, a file of that length is created. When a component takes a locks on such a memory region, the whole file is actually copied into its address space. Releasing the lock involves copying back the memory space to the file if the lock wasn't read-only. If the file is not visible through the file system, data transfer is done by collaborating PVM daemons.

PVM implementation based on Mach

If we want to use Mach NORMA IPC for implementing PVM in a cluster, the simplest approach is to replace UDP sockets by Mach ports, and the well known port naming scheme by a Mach nameserver. In this case, some already built-in PVM functions, like message sequencing, become redundant, but there is no real need to change higher levels of PVM initially.

PVM shared memory can be implemented in different ways, according to the degree of integration we want to achieve with the NORMA Mach primitives. This can be done by directly using XMM facilities, and relying on NORMA for memory coherency. In this case, the PVM daemons would have to be declared as pagers for their components. A component wishing to share memory with another component would ask its pager for memory object creation. It would then receive send rights for the object and would do a vm_map call. This send right would also be sent to the other intervening component (local or remote) which would also perform the mapping. The pager would declare this object as temporary, unless mapping to a file is needed for fault tolerance reasons.

If distributed process management with global process IDs is available, another solution to implement shared memory is to use the OSF/1 *shmget* system call. In this case, processes on other

nodes know about each other and can attach to already existing mapped regions.

PVM uses remote execution for starting daemons on all nodes, which in the generic case uses the Internet rexec/rshd facility. If we want to reimplement the way remote execution is done, we could use the *norma_task_create* call, to create processes on other nodes, like in OSF/1 MK-AD. However, in the initial prototype it will be acceptable if the initial creation of PVM daemons is done via rshd.

An evaluation of PVM has been done at Grenoble, and an initial port to OSF/1 MK has been done using its generic network transport (UDP sockets). The next step will consist in integrating Mach functionality as described.

PVM and Cluster Process Management

The current PVM architecture, where a PVM daemon runs on every node and acts as an agent for creating PVM components, synchronize them, etc... is well adapted to uncoupled systems where each node runs a separate OS entity. On a cluster environment however, this function is already realized by the collaborating Process Management Servers, that create an uniform process space across the cluster. We can thus consider another way of porting PVM to the cluster that will consist in a complete integration of its interface with the cluster OS Process Management. The PVM daemon will probably not be necessary, since the PVM interface could possibly be implemented in libraries using only cluster OS functionality, which would be linked with applications. An evaluation of the feasibility of this approach has to be done.

8.2 OSF Distributed Computing Environment

DCE is the OSF offer for building Distributed Computing Environments. The main DCE goal is to provide global interconnection between various types of Operating Systems and hardware platforms, by offering a standards compliant platform on which general distributed applications can be based. DCE consists mainly of two type of services:

- Fundamental Services, that provide basic tools, like Remote Procedure Call, Name Service, Security, Time Service and Thread support.
- Data Sharing Services, that provide Distributed File Access, Diskless Operation, MS-DOS file and printer support.

Highly Heterogeneous Environments

Achieving interoperability on highly heterogeneous environments implies several restrictions on functionality and on the way it is implemented, among which portability and independence from the underlying OS and hardware platforms are perhaps the more important. For these reasons, all DCE fundamental services are implemented using high level OS standard interfaces, and only Data Sharing services need extensions of the host system. On the other hand, functions like process migration, which are highly dependent of CPU homogeneity, are not part of DCE.

The Cluster is an Integrated Environment

On the contrary, the cluster environment described in this document has the opposite restriction of adapting as close as possible its distributed functionalities to the underlying system and hardware architecture in order to achieve a higher degree of integration and performance.

Thus, in our opinion, DCE and the cluster OS are not to be considered as competitors, since they are not designed to achieve the same goals. On the other hand, if the cluster OS is based on OSF/1 and supports the same application programming interface, it will support DCE. This means

that the relationship between DCE and the cluster OS will be the same as that which exists between DCE and with all platforms that support it: they should be able to interoperate.

Cluster OS and DCE Interoperability

We can consider cluster OS and DCE interoperability in several ways. The first level of cooperation is imposed by the goals of the cluster OS: in fact, if we consider the cluster OS primary goal, which was to extend the multicomputer programming model to networked workstations, we have to make the multicomputer accessible. This will be achieved by connecting the cluster to an external network (using the networking model described in Section 7) and providing a means for applications to access it as a network resource. The preferred way for accessing it will of course be DCE.

Accessing the cluster through DCE

Using DCE programming abstractions, applications can use the integrated cluster as a multicomputer, and submit CPU intensive jobs to it from any host located in an accessible network. The fact that the cluster is considered as another DCE host will make this access easier, since no special programming interface will be needed for general applications. On the other hand, applications wishing to directly exploit the cluster parallelism have the choice of using a parallel programing interface like PVM to do it.

DCE performance inside the cluster

Another level of cooperation can be achieved by offering, in the cluster environment, the same DCE programming abstractions. To achieve this, a Mach based RPC, as described in Section 6.2, should be usable inside the cluster by all DCE components. In this way, when applications which were built for heterogeneous environments are ported to the cluster environment, they should experience a considerable speed up. Integration of Mach based functions in other portions of DCE like the name service, is still an open question.

The cluster as a DCE cell

A third level of collaboration will be possible if the cluster file system is based on DFS. In this case, the cluster will be considered as a DFS sub-cell, and can thus be accessed from the outside world using the same access paradigms as for the rest of the DFS domain. To make this possible, a dual RPC implementation (or RPC translator) has to be provided in the cluster OS, that would use the generic DCE transport for communications outside the cluster, and the Mach IPC based transport internally.

9. Conclusion

In this document we have identified some of the problems raised by using the OSF/1 MK-AD system as the basis of an operating system for workstation clusters.

It is clear that, while this system serves as a useful model of a distributed architecture, significant modifications are required to provide a satisfactory operating system for clusters, since the latter must address issues such as dynamic configuration, distributed server cooperation, security, and node failure support.

We have outlined some possible solutions to these problems, but further analysis and prototyping work is necessary to evaluate these proposed solutions, and we hope that this will be carried out in cooperation with our collaborators in the Cluster Project.

We have also identified a number of areas where a relatively modest effort should lead to significant progress, including NORMA IPC enhancements, and adapting the PVM package to run over NORMA IPC. Other areas will require a longer term effort and resource commitments, as for example the distributed file system for clusters, and the development of an enhanced Mach IPC in the x-kernel framework. Finally, the area of process management has not really been addressed, other than to describe some available solutions, or partial solutions.

Finally, we want to thank everybody in the AD team who answered questions, proposed solutions and gave ideas. Without them, this document could not have been written.