Reliable computing in a UNIX United environment

K.H. Bennett, L.F. Marshall and B. Randell

Abstract
The Newcastle Connection is the name of a software subsystem which is added to physically connected distinct UNIX+ systems to form a coherent distributed computing environment. In this paper, the main features of the Newcastle Connection and its implementation are summarized. Ways in which this distributed UNIX system may be exploited to offer a more reliable environment than a single stand-alone system are then discussed. The main issue addressed is the potential to increase reliability and availability of the filestore, using replicate copy techniques. The generalisation of this concept to provide n-modular redundancy is then briefly outlined. Finally, a proposal to include primitives to support distributed atomic transactions is discussed. Throughout the paper, emphasis is placed on the adoption of sound structuring principles for design, requiring the recognition and clear separation of issues involved.

UNIX is a trade mark of Bell Laboratories.

Series Editor: M. J. Elphick

© 1985 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne,
Computing Laboratory, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

BENNETT, Keith, H.

Reliable computing in a UNIX United environment.

Newcastle upon Tyne: University of Newcastle upon Tyne,
Computing Laboratory, 1985.

(University of Newcastle upon Tyne, Computing Laboratory,
Technical Report Series, no. 201.)

Added entries
MARSHALL, Lindsay Forsyth. RANDELL, Brian.
UNIVERSITY OF NEWCASTLE UPON TYNE.

Abstract
The Newcastle Connection is the name of a software subsystem which is added
to physically connected distinct UNIX+ systems to form a coherent
distributed computing environment. In this paper, the main features of the
Newcastle Connection and its implementation are summarized. Ways in which
this distributed UNIX system may be exploited to offer a more reliable
environment than a single stand-alone system are then discussed. The main
issue addressed is the potential to increase reliability and availability of
the filestore, using replicate copy techniques. The generalisation of this
concept to provide n-modular redundancy is then briefly outlined. Finally, a
proposal to include primitives to support distributed atomic transactions is
discussed. Throughout the paper, emphasis is placed on the adoption of sound
structuring principles for design, requiring the recognition and clear
separation of issues involved.

+UNIX is a trade mark of Bell Laboratories.

About the author
Dr. K.H. Bennett is a Senior Lecturer in the Department of Computer Science
at the University of Keele, where he leads the Systems Research Group.
Dr. L.F. Marshall has been a Research Associate at the Computing Laboratory
of the University of Newcastle upon Tyne since October 1981.
Professor B. Randell has been a Professor of Computing Science in the
Computing Laboratory of the University of Newcastle upon Tyne since 1969.

Suggested keywords
ATOMIC TRANSACTIONS
DISTRIBUTED SYSTEMS
NETWORKS
NEWCASTLE CONNECTION
RELIABILITY
SYSTEM STRUCTURING
UNIX

Suggested classmarks (primary classmark underlined)
Dewey (18th): 001.64404
U. D. C. 519.687
Reliable Computing in a UNIX United Environment

K. H. Bennett, L. F. Marshall* and B. Randell*

Department of Computer Science,
University of Keele,
Keele, Staffs., ST5 5BG

Abstract

The Newcastle Connection is the name of a software subsystem which is added to physically connected distinct UNIX* systems to form a coherent distributed computing environment. In this paper, the main features of the Newcastle Connection and its implementation are summarized. Ways in which this distributed UNIX system may be exploited to offer a more reliable environment than a single stand-alone system are then discussed. The main issue addressed is the potential to increase reliability and availability of the filestore, using replicate copy techniques. The generalisation of this concept to provide n-modular redundancy is then briefly outlined. Finally, a proposal to include primitives to support distributed atomic transactions is discussed. Throughout the paper, emphasis is placed on the adoption of sound structuring principles for design, requiring the recognition and clear separation of issues involved.

*Address: Computing Laboratory, University of Newcastle upon Tyne, Newcastle upon Tyne NE1 7RU.
+UNIX is a trade mark of Bell Laboratories.
1. Introduction

The Newcastle Connection[1] is a software subsystem written at the University of Newcastle upon Tyne which may be added to each of several stand-alone UNIX[2] systems which are physically interconnected. The resultant distributed system is known as UNIX United, and can be used as if it is a conventional UNIX system. From the user, or user program viewpoint, this is achieved without any apparent change either at the command level (the shell) or at the kernel level (the system calls). For example, a remote file may be opened and accessed in exactly the same way as a local file. Details of the intersystem communication mechanism and protocols are not visible. A corollary is that modifications are not required either to user source programs or to the UNIX operating system programs which reside above the kernel (we shall collectively call these two classes of program the "application level" programs). Further details of UNIX United are given in section 2.

One of the often-claimed advantages of a distributed computing system is that the redundancy which typically exists in such a system can be exploited to provide increased reliability of computing service. In section 3, work is described which provides reliable file storage, relying upon the Newcastle Connection to handle all issues of distributedness. The objective is to provide a more reliable environment which is functionally equivalent to the unreliable distributed environment. There are two principal components in the implementation: a naming layer, which maps the names of reliable, virtual files onto the names of several component copies; and algorithms to manage the consistency of the several copies in the face of failure.

In section 4, the addition of a set of atomic transaction primitives to UNIX and UNIX United is discussed. This necessarily augments the functionality of a single UNIX system. However it is of interest to investigate how a facility which many would regard as needed even in a centralised UNIX system, may be provided within the UNIX United structure.

2. UNIX United

Naming

A UNIX United namespace is aggregated from the namespaces of the component systems. It can be envisaged as the mounting of the complete name space of one UNIX system in the name space of another. Files, directories and devices are named (for a given user) relative to either of two movable context names, the current working directory and the current root directory (in the usual UNIX manner). In Figure 1, unix1, unix2 and unix3 are the names of the root directories of stand-alone UNIX systems, as they appear in the context in which they are mounted.
If the root directory `/` is at `unix2`, the file `myfile` may be copied to the directory `brian` by executing the standard UNIX `copy (cp)` command:

```
cp /usr/keith/myfile ../unix3/usr/brian
```

No new facilities or conventions are introduced to name or access a remote file. The standard UNIX notation `".."` which identifies the parent directory of a file or directory enables us to walk round the whole name tree (assuming we have permission). The current working directory may be placed as usual by e.g.

```
cd ../..unix3/usr/brian
```

The file `bfile` may be copied to directory `keith` by:

```
cp bfile /usr/keith
```

Note that the code executed for `cp` in both the above cases is held in `unix2/bin/cp`. This is because a shell process conventionally locates the code of a command relative to its root, which has not been altered in the above example. There is no implication that the code of `cp` must actually be executed on the physical CPU corresponding to `unix2`; that decision is not visible at the application level.

In recent years, the designs of a number of distributed computer systems based upon the UNIX operating system have been reported in the literature; see for example [3,4,5,6,7,8,9,10]. We argue that UNIX United is characterised by its adherence to what is termed the principle of "recursive functionality". This means that the distributed Unix United system is functionally equivalent to a single, centralised UNIX system. All the standard functions of UNIX are applicable without change within the distributed system. A major requirement in a recursively structured system is that all pathnames are relative [11,12]. A system based on absolute names would conflict with the ability to extend the
namespace without modifying applications programs. Some distributed UNIX systems employ a single system-wide root, positioned at the base of the tree, which provides a global context for component UNIX system names. The namespace at this level may only be expanded by guaranteeing that there are no name clashes. In UNIX United, today's base directory is tomorrow's subdirectory in some larger namespace. It is thus evident why UNIX has proved a suitable vehicle for providing a recursively structured distributed system. The hierarchical naming scheme for files, directories and devices allows namespaces to be combined without name clashes. The construction process of building a UNIX United namespace from component UNIX (United) namespaces potentially can be carried on indefinitely, resulting in very large distributed systems connected (in practice) by a variety of networking technologies.

Management issues

The standard UNIX mechanisms for file protection and controlled sharing of files apply without change in UNIX United. Each component system will have its accredited list of users and groups, its own manager (superuser), accounting and so on. The manager will additionally be responsible for allocating access rights for remote users. This is achieved by setting up a mapping table of (username, remote system) into a local user name. The remote system is named as usual by a relative pathname.

Implementation

UNIX United is implemented by imposing an extra layer of software between the kernel and the application level programs on each participating site (Figure 2).

```
level 2
   Applications programs

level 1
   Newcastle Connection

level 0
   UNIX kernel
```

Figure 2

The UNIX kernel provides a modest number of system procedure calls. The Connection intercepts those that involve files or devices; those to be handled locally are passed unchanged to the local kernel. Those to be handled remotely are passed to the Connection on the appropriate system (plus additional information such as the user identifier). The Connection layer is normally implemented as a library, and programs using systems calls must be relinked to this library (programs not so linked are hence purely local). However some versions of UNIX United have been constructed by installing the Connection within the kernel, hence obviating the need for relinking.

The inter-machine communication mechanism is based upon a remote procedure call [13]. The call provides an "exactly once" semantics, based on non-volatile sequence numbers. This still permits orphan processes if the client system crashes. Currently, such orphans are intended to be dealt with at a higher level which supports atomicity in user programs (section 4), though the Connection does carry out some orphan killing without supporting atomicity.

On a remote system a spawner process, which runs continuously and has a well known address, sets up a fileserver process on behalf of the user process.
Subsequently, the user process deals with this server directly, using the name returned by the spawner. When a file is opened, the Connection on the client machine makes an entry in a per-process table indicating whether or not the file descriptor (an integer used to refer to the file between open and close system calls) refers to a local or a remote file. The table also contains routing information.

The fileserver on a particular system interprets the fragment of pathname local to it and then (using a remote procedure call) passes the remainder to a server on the next system. (It is planned to make it possible, as in the example in Section 2, for a pathname to a remote file to include more than one UNIX system.) These servers form an extension of the user's environment, so that any changes in the environment must be reflected by them. When a process "forks" (duplicates itself) its remote fileservers must also fork.

Issues of network interfacing are more fully discussed in [14]. Although the first versions of the Newcastle Connection were closely integrated with specific network drivers, the basic approach now being explored is to implement the remote procedure call in terms of a uniform datagram service (UDS). This provides a uniform programming interface to different types of network services, without attempting to provide the abstraction of a single internetwork. Remote nodes are addressed in the UDS by a (network, host, port) triple. An adaptor subsystem maps the UDS level into the various interfaces provided by the network services (X25, Ethernet, Cambridge Ring, dial-up, etc.).

3. Reliable filestore

Introduction

In most computer systems, the user view of the filestore (its reliability, security, performance, etc.) is critical to the success of the system as a whole. This is no less true in a distributed system than a centralised system. The reliable storage of data is crucial to many organisations, for whom the cost of keeping replicate copies, so as to achieve the level of reliability required, is a small price to pay in terms of current disc costs. In this section, we shall concentrate on a more stringent requirement, data availability, defined as the probability that data is not just held reliably, but is on-line and accessible. We shall describe a project undertaken at the University of Keele to explore the improvement in availability by storing replicate file copies in a distributed environment. Many of the ideas were established in a pilot project described in[15] and [16]. An implementation was then undertaken using UNIX United as the environment [17], and it is the latter which will be described.

Naming

In the Keele system, a single, system-wide naming tree is presented to the user. A single root directory exists, which is replicated on all participating systems. Subdirectories and files are not necessarily replicated on all systems. The set of systems on which a child directory or file is replicated must be a subset of the systems on which the parent directory is replicated. The appearance and disappearance of individual UNIX systems is evident, not by changes in the namespace, but by changes in the availability of files.

A typical name space is shown in Figure 3, where the notation x(u1, u2) means that a file or directory x is replicated on UNIX systems u1, u2.
Each UNIX system contains the full pathname from root to any files stored on it. Therefore, if a given system is online and available, all files with copies held by it are potentially accessible.

The actual availability depends on additional factors such as the availability of the communications network, the pathname used, and the probability that the filestore on a UNIX system may be partitioned over several volumes. A simple model of the availability, where \( p \) = probability that a UNIX system is operational, is:

\[
1 - (1 - p)^n
\]

for an \( n \) machine system. Hence greater availability is achieved because the UNIX systems are independent. The availability would in contrast be degraded if the successful interpretation of a pathname required several independent UNIX systems to be on-line.

**Implementation**

A "replication" software subsystem is interposed above the Newcastle Connection, but below the applications programs. System calls involving files are intercepted; if they do not use replicated files, they are passed to the Connection with only a name mapping. If the call is an operation on a replicated file, the layer maps the name onto component copies, and carries out consistency management. The facilities of the Connection are used to treat local and remote copies identically.

The replication layer provides a mapping from a single virtual pathname to one or more pathnames of the copies. As long as all application level programs are linked to the replication layer, there are no name clashes or ambiguities. An arbitrarily complex mapping could be used, but currently a very simple scheme is used. The virtual pathname is used unchanged as the postfix of each copy name. On a particular system, the prefix is the relative pathname from the local root to the root of the remote system holding the copy. This direct mapping means that the handling of single copy files is trivial.

The majority of the code in the replication layer is concerned with the management of the consistency between file copies. The requirement of multiple copy techniques could be to achieve total consistency i.e. all copies are identical and up to date at access. This decreases availability because not all copies may be on-line. In our implementation, a majority voting strategy is used, where only a majority of up to date copies must be available before an update can proceed. This means that successive updates to the same user file always
have at least one file copy in common, and a local version number can be used to identify consistent copies. We must then be able to cope with off-line copies which subsequently become available. During a multiple copy update, the copies may become temporarily inconsistent, but this must not be visible to the user. Additionally, crashes can occur at any time in the participants. There may be a majority available at a file open, but not at subsequent operations on the file. The algorithms used are based on a centralised synchronisation, distributed recovery scheme originally described by [18]. A coordinator site is appointed to manage an update (say) operation. If this site crashes, a new coordinator is elected. Sufficient state information is stored so that the recovery manager on a crashed site can establish a well-defined state on reboot.

Developments

The requirement that a parent directory must be replicated on at least those systems on which its children are replicated might seem to be in conflict with the potentially unbounded UNIX United namespace. However it is envisaged that one (or more) subtrees within a larger UNIX United namespace could be used to offer a highly available filestore [19]. Physically, the subtree could correspond to several UNIX machines linked by a high speed local area network. Functionally, the "reliable subtree" is indistinguishable from other parts of the namespace. Files within it may be accessed from outside, and vice-versa, in the standard way. Of course, due consideration must be paid to reliable communications, to achieve high availability to remote users.

In addition, experiments have been undertaken at Newcastle on the use of Triple Modular Redundancy to mask failures at the UNIX system level. The latest plan[20] is to build a TMR (triple modular redundant) virtual node inside a UNIX United system. Functionally, the resulting system is still equivalent to a conventional UNIX system, and the user accesses it in the normal way. However, a process executed within a TMR node will actually be executed in triplicate (and preferably in parallel), and the results voted upon. Thus hardware crashes, etc., can be detected at voting time, and the faulty component automatically switched out. Subsequently, another component may be substituted for it. Files within the TMR node are stored in triplicate. The TMR node is implemented by inserting a Guide layer and a TMR layer between the applications level and the Newcastle Connection. The guide layer intercepts kernel calls, and manages the triplication. Its principal functions are to provide a naming mechanism, and to coordinate the voting when triplicated file servers are set up via the Connection layer of each system. The main function of the TMR layer is to manage the triplicated execution of user processes, including the handling of signals. It can use the facilities provided by the Connection to handle all matters concerned with the distribution of files, processes and so on.

4. Fault tolerance in distributed systems

Figure 4 summarizes the structuring approach that has been developed at Newcastle for designing and describing fault tolerant systems, and that is fully explored in [21].
A generalised fault tolerant component is presented with service requests. If they are outside the specification, an interface exception is raised to the calling level. Internal exceptions, or exceptions from subcomponents will force entry to code to handle abnormal activity. It may be possible to mask the exception (e.g. by backwards recovery), and return to normal operation. If not, an exception is raised in the calling component. The model is inherently recursive.

The principle of recursive functionality should apply not only to normal behaviour, but also to exceptional behaviour. A fault tolerant distributed system should be functionally equivalent to the fault tolerant component systems of which it is comprised. Thus the Newcastle Connection reports exceptions in the same terms as those used by the UNIX kernel, and not in terms concerned with distribution. Within the Connection itself, however, attempts are made to recover from faults which specifically arise from the distribution. The remote procedure call, for example, tries to recover from lost and duplicated messages, and to implement an exactly once semantics. It cannot recover from faults which derive from the user level; an example is the need to execute a group of commands either completely, or not at all.

A form of fault tolerance that is sometimes provided in both centralised and distributed systems is the atomic transaction. An atomic transaction is composed of one or more actions, the transaction having the following properties:

P1: The transaction is executed entirely, or not at all (failure atomicity).

P2: If two or more concurrent transactions access a shared resource, the effect is as if the transactions had been obeyed strictly sequentially (serialisation atomicity).

The usual versions of UNIX do not support transactions and hence neither does UNIX United. However, it is of interest to consider how primitives to support them might be added, and to explore the implications for structuring. The first requirement is to add recovery primitives to a centralised UNIX system. This could be done by augmenting the kernel, or by adding a subsystem immediately above the kernel. The three system calls proposed for this purpose in[11] have
been implemented at Newcastle earlier in the distributed recoverable filestore described in [22].

They are:

1. Establish Recovery Point (erp): start state-saving and locking
2. Discard Recovery Point (drp): discard saved state and release locks
3. Restore Recovery Point (rrp): go back to state at recovery point

The erp operation returns an integer for use in the other calls. The act of restoring to a recovery point amounts to rolling back the execution. Discarding a recovery point is a commit operation, because we can no longer return to the saved state at the recovery point. It is envisaged that the three primitives will be augmented in UNIX with a fourth "Prepare to discard recovery point".

We have seen that in UNIX United, a process may be distributed (via servers) over several sites. Any or all of these sites, including the initiator, may fail at any time. Since the Connection deals with issues of distributedness, it must now provide tolerance to this type of fault but in a distributed environment, in order to preserve the functionality of the underlying system calls provided to support fault tolerance. One useful technique for implementation is the two phase commit protocol [23]. It is possible that a single UNIX system could be implemented on a multiprocessor configuration which can exhibit partial failure. This could require primitives to support two phase commit on a single system, and hence the fourth system call is needed.

5. Conclusions

The implementation of the Newcastle Connection began in late 1981, and it was in operational use by mid 1982. Since then, it has been installed at a number of sites worldwide, on a variety of equipment, and using a variety of network technologies. The Keele filestore, and a partial prototype of the TMR node, were implemented subsequently. The proposal to add atomic action primitives has yet to be implemented within UNIX United. The theme running through this paper has not been implementation techniques, however, but the importance of proper structuring, and in particular the adoption of two recursive structuring principles in design:

a) a distributed computer system should be functionally equivalent to the component systems of which it is comprised.

b) fault tolerant systems should be constructed from generalised fault tolerant components.

The adoption of the first principle has resulted in a distributed computer system in which all issues of networking, protocols, etc., are invisible to users and their programs. The design has stressed the logical separation of distinct issues, such as distributedness, atomicity and TMR, and their implementation in equally distinct layers.

In terms of a distributed system, the second principle requires that a fault tolerant distributed system should be functionally equivalent to to the generalised fault tolerant component systems of which it is comprised. This provides a methodology for the designer for achieving fault tolerance in a well-structured manner, with strong constraints on where exceptional events should be handled.

Thus the main result from the work described in this paper is the production of further evidence in support of the recursive structuring principles that have
been used in the design not only of the Connection, but also for other UNIX-based enhancements such as replicated files, atomic transactions and the TMR node.

Acknowledgements

A number of other people at Newcastle have made important contributions to the work described here, including Dave Brownbridge, who with Lindsay Marshall was responsible for the first implementation of the Connection, Fabio Panzieri and Santosh Shrivastava, who developed the Remote Procedure Call Protocol which this implementation used, and Li-Yi Lu, who worked on Triple Modular Redundancy techniques. The Keele distributed filestore was implemented by Pearl Brereton. Valuable contributions were made by Paul Singleton and also Ken Lunn, who originated the idea of the "overlay mount" technique. The work at both Keele and Newcastle was funded by the UK Science and Engineering Research Council.

References


