Open Software Foundation **R. Zee (DEC)**
Request For Comments: 69.0
November 1994

DCE 1.2 CDS FUNCTIONAL SPECIFICATION

1. INTRODUCTION

In keeping with DCE Release 1.2's stated goal of expanding deployment (see [RFC 63.0]), the changes to CDS mainly address the robustness of the CDS server to address the scalability issue. These changes will significantly reduce the use of virtual memory at a server that is receiving many updates. The user will benefit from a more stable and reliable service.

The major change is the replacement of the existing server backend (B-tree implementation) with an alternate tree implementation (one of AVL, 2-3 tree, B+-tree, Red-Black) which is more manageable and less prone to fragmentation over time, thereby accomodating larger and more robust clearinghouses.

Additional work includes any changes to support single-threaded RPC client work and incomplete DCE 1.1 functionality (integration changes needed for hierarchical cells or transitive trust).

2. TARGET

This technology is primarily to be used by cell administrators and system managers as an improvement in robustness of the CDS server. A secondary user would be all CDS applications because of better availability of the CDS servers.

3. GOALS AND NON-GOALS

The *goals* for the DCE 1.2 CDS work are:

(a) Significantly reduce the server's use of virtual memory at a server that is receiving many updates. This should not induce major code changes in other subcomponents of CDS, and no code changes in the other DCE components.

(b) In reimplementing the CDS server database, if an in-memory implementation is chosen, it will have comparable performance in name lookup queries.

(c) Provide necessary tools with this new database implementation to effectively dump and salvage the database (exact extent of functionality is TBD).

(d) Preserve existing CDS server communications behavior relative to CDS clerks and other CDS servers (no server-external protocol changes are expected).

(e) Support single-threaded RPC client work by making appropriate changes to the CDS library.

(f) Adhere to established coding practices for portability, I18N, and serviceability.

The *non-goals* for the DCE 1.2 CDS work are:

(a) At this time, we have not committed to deliver a single process CDS clerk per system for DCE 1.2.

(b) It is not expected that any CDS work is necessary to support XFN work for DCE 1.2.

4. TERMINOLOGY

The term ''database'' refers to a CDS server clearinghouse (typically one clearinghouse per server system). This means both representations of a clearinghouse on a running CDS server system — (1) the running, in-memory representation and (2) the clearinghouse file on disk used as the backing store mechanism during checkpointing.

5. REQUIREMENTS

In investigating how to reimplement the CDS server database, we are considering the following factors (prioritized):

(a) Appropriate for CDS usage paradigm (variable length keys, enumeration, etc.).

(b) Performance (both in virtual memory usage and CPU time).

(c) Scalability (larger clearinghouses).

(d) Code maintainability (robustness, clarity of form).

(e) Code shareability (prefer to use existing code, but not by sacrificing other factors).

We also plan to reduce the duplication of information at a server and will more aggressively purge stale data.

6. FUNCTIONAL DEFINITION

6.1. Reduce CDS Virtual Memory Usage

Task 1: Reducing the CDS server's use of virtual memory. In improving CDS server usage of virtual memory, we break this down into the following sub-tasks:

(a) Sub-Task 1: Replace `db_btree.c` with a more efficient mechanism as defined by the requirements above.

This will largely be done by replacing the existing CDS database implementation with a more manageable (code-wise) implementation that is less prone to fragmentation over time. We are investigating using the existing security 2-3 tree code, the db44 B+-tree code, an AVL tree implementation from Berkeley, and a red-black tree implementation from Project Pilgrim (University of Massachusetts).

(b) Sub-Task 2: Store fullnames as cell relative in CDS attributes, some RPC attributes, and user-defined `VT_fullname` attributes.

Dependency: Commitment from RPC not to change their encoding of CDS fullnames in their RPC attributes (currently, the RPC attributes store the fullname as a `VT_byte` and encode it in little endian). Affected RPC attributes are `RPC_profile` and `RPC_group`.

The advantage for CDS attributes is speed up of propagating name changes. For user and RPC atributes, we automatically keep them up to date without administrative overhead. This will require a conversion on startup and code to remove cell names from input and add cell names to output.

(c) Sub-Task 3: Make use of the fact that the cell root directory is in all clearinghouses and build replica pointers on the fly.

At each server, this will require a new table to map clearinghouse UUID's to their towers, which will need to be updated if a clearinghouse's address towers change. All replica pointers and child pointers will internally store only the clearinghouse UUID. On output, this table will be referenced, and the replica pointers will be built dynamically.

(d) Sub-Task 4: Modify `sets_lib.c` so that unnecessary history is removed prior to the skulk.

There are two types of attributes: `AT_single` and `AT_set`. All attributes are stored in the `DBSet_t` data structure. In order to keep replicas up to date, enough state from past transactions must be maintained so that replicas can build the current state of an attribute at any given time (replicas may not have seen all updates because of failed propagations). After a skulk, we can remove old state because only then do we guarantee that the replica has seen all of the updates.

Since members of a `DBSet_t` are added in timestamp order, after a set member is inserted, we can update the state of other set members because updates may invalidate earlier updates. Prior to 1.2, CDS kept all transactions and did not clean out invalidated set members until after a skulk. Practical experience has shown that application servers are doing RPC export/unexport operations on startup/shutdown, which generates many updates of the same data to the namespace. Application server entries can grow quite large, especially if skulks are not completing successfully. It turns out that not all of the past transactions that have been invalidated need to be maintained. The server only needs to keep certain last values, thus enabling the server to remove many past transactions dynamically.

(e) Sub-Task 5: Miscellaneous virtual memory usage cleanup in server.

Sprinkled throughout the server code, we will be investigating where we can reduce or eliminate memory usage in data structures and routines. Fullnames and large buffer usage will be verified.

(f) Sub-Task 6: Backwards compatibility of new database implementation with previous 1.1 B-tree implementation.

Other components of the server (Transaction Agent, Background, management) sometimes used various optimizations in calling the previous database code. We will attempt to preserve this on a case-by-case basis.

6.2. Salvager

Task 2: `cds_salvage_db`-like tool, exact functionality is TBD.

Dependency: On Task 1 above to establish data structures.

The tool is expected to be able to dump the database, detect database corruption, and allow the user to fix or modify portions of the checkpointed, on-disk clearinghouse file.

6.3. Single-Threaded Client

Task 3: Single-threaded client implementation of the CDS API to support single-threaded RPC client work in 1.2.

Dependency: Input from the RPC developers will determine if the single-threaded CDS library is a separate shareable `libdce.so` or not.

The pre-1.2 implementations of the CDS library have multiple threads for reads and writes to the CDS clerk process. We will change this so that the library only does read and writes to the clerk in a single thread while caching its sockets.

7. DATA STRUCTURES

As all of the CDS server changes are internal, no data structure changes should be visible to the user. Note, the structure of the clearinghouse file may change with the new database implementation. Since this format is also hidden from the user, nothing will be exposed.

8. USER INTERFACES

The only user-visible aspects of this work will be tools supplied to analyze, dump, and repair (simple, localized data corruption) the server database. The exact interface is somewhat dependent on the implementation chosen, however, a utility with options is the envisioned result.

9. API'S

No new user-visible API's will be supplied with this work.

10. REMOTE INTERFACES

No new remote interfaces will be supplied with this work.

11. MANAGEMENT INTERFACES

At this time, the only possible addition is a tool to convert an existing DCE R1.1 or earlier CDS clearinghouse file to the new format, if there is one. We are currently investigating the possibility of preserving the existing clearinghouse file format to eliminate the need of a database converter tool. If a new file format is chosen for the database, the existing `dce_config` management tool will be modified such that when the CDS server starts up and detects the old format, the image will exit with a specific status. `dce_config` will then ask the user if he/she wishes to convert the CDS clearinghouses at this node and invoke the converter tool appropriately.

12. RESTRICTIONS AND LIMITATIONS

If an in-memory implementation is chosen, the CDS server will still be a memory-hungry daemon for larger clearinghouses. However, through improved memory usage, the bar should be raised as to how large a clearinghouse can get before it affects the rest of the server system.

13. OTHER COMPONENT DEPENDENCIES

Dependency: We would like to have discussions with HP on possibly sharing code with their security database backend. We need to weigh the pros and cons of an all in-memory implementation vs. on-disk (and what to cache). This should be done within the next few weeks.

14. COMPATIBILITY

The DCE Release 1.2 version of the CDS server will be wire-compatible with all previous versions of CDS clerks and servers. No wire protocol changes will be made.

15. STANDARDS

The DCE Release 1.2 version of the CDS server will adhere to established coding practices for portability, I18N, and serviceability.

16. OPEN ISSUES

(a) A tree implementation has not yet been chosen to replace the existing 1.1 CDS server backend.

(b) Functional definition of the database dumper/fixer tool.

REFERENCES

[RFC 63.0] R. Mackey, J. Pato ''DCE 1.2 Contents Overview'', October 1994.

AUTHOR'S ADDRESS

Roger Zee
Digital Equipment Corporation
550 King Street, LKG2-2/Z7
Littleton, MA 01719
USA

Internet email: zee@tuxedo.enet.dec.com
Telephone: +1-508-486-5288