Open Software Foundation **J. Gait (Transarc)**
Request For Comments: 89.0
December 1995

A BULK STATUS RPC FOR DFS

1. INTRODUCTION

This RFC describes a new DFS RPC to return the status of files in a directory in bulk form, rather than one file at a time. The bulk status RPC is designed to largely replace the lookup RPC by doing the work of many lookup RPCs in one RPC. The motivation is to improve the throughput, measured in files per second, of directory browsing by reducing the number of RPCs required to browse directories. Directory browsing in DFS currently requires at least an RPC (to lookup the status) for each file in the directory. For ''small'' directories (where the definition of ''small'' should incorporate most ordinary working user directories), the bulk status RPC browses the entire directory in one RPC. The bulk status RPC does not reduce the amount of work done to browse the directory; both server and client do precisely the same amount of work and the same data travels over the wire. However the efficiency introduced in the bulk status RPC by reducing the RPC count may potentially improve the throughput of directory browsing by a significant amount (say, up to 50%).

1.1. Bulk Status and DFS Semantics

DFS semantics allow an RPC to get in the way of other DFS operations by causing cascaded RPC activity and by revoking tokens needed to make progress. Thus, a DFS RPC is competing for system and token resources to complete a task that is necessary for the operation of DFS.

The bulk status RPC does not fit this picture. First, the bulk status RPC is intended to reduce the RPC count, so it should not induce cascaded RPC activity. Second, the bulk status RPC is not required by DFS semantics, so it should not interfere with the progress of any RPC that is required by DFS semantics, e.g., by revoking tokens or by consuming tokens in an uncontrolled way.

The bulk status RPC is comparatively heavy-duty in that it runs far longer than any other RPC. It seems right to introduce mechanisms to prevent the bulk status RPC from exceeding the already unbalanced execution time. Thus the bulk status RPC should not retry, either on the client or on the server. It follows that the server should not seek to re-obtain a lost directory token. Instead, the bulk status RPC is designed to fallback gracefully to the conventional lookup RPC when necessary.

In line with the above discussion, clients that support the bulk status RPC should not start superfluous bulk status RPCs to servers that do not support the RPC. At steady state clients should only request bulk status service from servers that provide it, otherwise lookup service is employed on a per-file basis.

1.2. The Mechanics of Retrieving Status in Bulk

In general it is not feasible (because space to store the information is limited) to retrieve the status information for all the files in a directory in one RPC (or even for all the files in a single chunk of the directory), so a fixed window of files (defaulting to `AFS_BULKMAX`) is bulk stat-ed on each RPC. The value of `AFS_BULKMAX` should be large enough to incorporate all the files in most user directories, but small enough to limit the amount of system and token resources engrossed by the call. The window is moved through the directory so all files are eventually stat-ed. The bulk status RPC is expressly designed to support directory browsing. Directories are normally browsed from beginning to end, so the file currently being looked-up is understood to be in the window currently being stat-ed, as long as the bulk status RPC is in the path of a directory browser or the directory is small enough. So, to belabor what is by now self-evident, and for verisimilitude supposing `AFS_BULKMAX` = 32, a lookup of the first file in the directory actually collects the status of the first 32 files in the directory, with lookups hitting the caches thereafter, until the 33rd file, which results in collecting the next 32 files, and so on.

The catch is that not all directory access is by directory browser. Any other than a beginning to end parse of the directory defeats the bulk status optimization, unless the directory is small enough to be retrieved entirely in one bulk status RPC. Some examples of directory browsing are `ls -l directory`, `find` and `du`; the bulk status

RPC improves the throughput of these activities. But `ls -l file` may cause a bulk status RPC for a window of files that does not include the specified file. Accordingly, there should be a provision to turn off bulk status support on both the client and the server, so system administrators have control over usage patterns that break bulk status. The decision depends on the mix af applications on the client and in the enterprise, and on the style of interactive use prevailing at the site.

2. GENERALIZED STATUS INFORMATION

In DFS the lookup (status) RPC returns the following information for a single file:

(a) The file identifier, or fid.

(b) The status of the file.

(c) A token that at least allows read status for the file, and may allow much more.

The lookup RPC passes the name of the file to the server, and processes a returned error code. When the lookup RPC succeeds, the client cache manager uses the name of the file as a key to install the fid in the client name cache, and uses the fid as a key to install the status and token in the client status cache. Subsequent invocations of lookup find the required information locally until either the token is lost or the caches are purged of the entry.

A client that invokes the bulk status RPC does not know in advance the names of the files whose status will be returned and must be prepared to process per-file errors even though the RPC may have succeeded. Accordingly, the bulk status RPC returns the following information for a window of files in a directory:

(a) The name of each file.

(b) The corresponding fid.

(c) The corresponding status.

(d) A token for the file that at least allows read status.

(e) An error code that is set by the file server on a per-file basis.

This is the **generalized status** information for the file.

If the server returns an error for any file processed in the bulk status RPC, the the client cache manager omits the file when generating information for the name and status caches.

In case the bulk status RPC itself returns an error, the returned window of generalized status is not processed into the name and status caches and there is no retry: the client falls-back on the lookup RPC.

2.1. Structure for Generalized Status

An instance of the following structure is returned by the server to the client for each file being stat-ed:

```
typedef struct bundledStat
{
    afsFid fid;
    afsFetchStatus stat;
    afsToken token;
    error_status_t error;
} bundledStat;
```

In case an error occurs on the server while obtaining any of the fid, status or token for the file, the error is set to an appropriate value that indicates the character of the error. (The file name is returned separately.)

2.2. Conformant Array for Generalized Status

The ensemble of generalized status is returned by the server to the client in a conformant array. (See the OSF DCE Application Development Guide, Rev 1.0, 17.14.5 for a discussion of conformant arrays. Put simply, a conformant array is sized at run time, rather than at compile time.)

```
typedef struct bulkStat
{
    unsigned32 bulkStat_len;
    [length_is(bulkStat_len)] bundledStat
                              bulkStat_val[AFS_BULKMAX];
} bulkStat;
```

The conformant array only returns generalized status for precisely the files that have been stat-ed on the server. The value of `AFS_BULKMAX` is the default maximum size of a directory window.

3. FORMAL DEFINITION OF BULK STATUS RPC

Here is a formal IDL definition of the bulk status RPC:

```
error_status_t AFS_BulkStatus (
    [in]        handle_t        h,
    [in]        afsFid          *dirFidp,
    [in]        afsHyper        *offsetp,
    [in]        unsigned32      size,
    [in]        afsHyper        *minVVp,
    [in]        unsigned32      flags,
    [out]       bulkStat        *bulkStats,
    [out]       afsHyper        *nextOffsetp,
    [out]       afsFetchStatus  *outDirStatusp,
    [out]       afsToken        *outTokenp,
    [out]       afsVolSync      *syncp,
    [out]       pipe_t          *dirStream
);
```

The parent directory and current offset are specified in `dirFidp` and `ffsetp`, while the server returns a new offset in `nextOffsetp` The server returns status and a token for the directory in `outDirStatusp` and `out-Tokenp`. The client informs the server of its buffer size in the `size` parameter and passes various flags in the `flags` parameter. The `minVVp` and `syncp` parameters are technically related to the storing fileset. The generalized status is returned in two parts. The file names are returned in directory wire format via the `dirStream` pipe and the remaining status is returned in the `bulkStats` conformant array.

4. THE CLIENT ARCHTECTURE

The bulk status RPC is insinuated into the lookup code path in `nh_dolookup()`. The parameters to the lookup call are the directory and the name of a file in the directory. The fid for the file is returned as an out-parameter. Following is a sharply truncated pseudocode version. The `if(bulkStatFlag){}` clause is the new code added in `ns_dolookup()` to support bulk status.

```
ns_dolookup(dscp,namep,...)
{
    int bulkStatFlag=1;

redo:
    /* acquire directory token */

    cm_GetTokens(dscp);

    /* access name cache */

    code = nh_lookup(dscp, namep, ...);

    if (!code)
        return 0;
```

```
        if (bulkStatFlag)
        {
            if (dscp->opens > 0)
            {
                /* bulk status RPC */

                cm_BulkFetchStatus(dscp);

                bulkStatFlag = 0;
                goto redo;
            }
        }

        /* lookup RPC */

        code = AFS_Lookup(dscp, namep, ...);
        return code;
    }
```

There are a number of points to make about this pseudocode representation:

(a) The bulk status RPC is invoked in case the directory is open and a file is being looked-up from the directory.

(b) If the file being looked-up is found in the name cache, then no RPC is necessary from this code path.

(c) The bulk status RPC is invoked at most once for each call to `ns_dolookup()`. This is enforced by the treatment of `bulkStatFlag`.

(d) The bulk status RPC is an order of magnitude more time-consuming than the lookup RPC, so a directory token is re-acquired when the RPC returns.

(e) There has been time for some other thread to install the file in the name cache, so even if the RPC fails, the name cache is re-accessed.

With the inclusion of the bulk status call, there are four possible paths through `ns_dolookup()`:

(a) The first path finds the file in the name cache and thus returns without making any RPC.

(b) The second path does a bulk status RPC and then finds the file in the name cache.

(c) The third path does a bulk status RPC, still fails to find the file in the name cache, so also does a lookup RPC. This may occur if there is a token problem or if the file being looked-up is not in the current window.

(d) The fourth path enters the bulk status code path, aborts for some reason (e.g., the server does not support bulk status) in `cm_BulkFetchStatus()` before attempting the bulk status RPC, then falls through to conventional lookup code paths.

Thus this codepath may result in 0, 1 or 2 RPCs depending on circumstances.

4.1. Directory Windows

A **directory window** is a sequence of `AFS_BULKMAX` or fewer files in directory lookup order. The beginning of the window is determined by `offsetp` and the beginning of the next window is signaled by the server to the client in `nextOffsetp`. The maximum size of the window is set by the server. The default maximum value is `AFS_BULKMAX` files (set in the IDL file that defines the RPC), however the system administrator may reset this value to a smaller number at runtime.

4.2. Token Handling

It may be that a particular client invoking the bulk status RPC does not desire the corresponding tokens to be returned as part of the generalized status. In this case the client may set a bit in the `flags` parameter that instructs the server to skip the token code altogether.

DFS RPCs normally verify that the client still has the necessary token when the actual RPC has completed, since it is always possible that a token has been lost while the RPC was in-progress. The probability of losing the directory token during the bulk status RPC is much higher than it is for other RPCs because it runs over ten times longer. Thus the client-side bulk status RPC makes the check for lost token and aborts if necessary, thereby falling through to conventional lookup behavior.

4.3. Disabling the RPC

For benchmarking purposes it is convenient to be able to disable the bulk status RPC at runtime on a per-client basis. Further, the usage pattern of a particular client may not be favorable for bulk status. By default bulk status is enabled at each client, however the system administrator may disable bulk status on a particular client at runtime.

4.4. Server Does Not Support Bulk Status

The success of DFS is largely predicated on pervasive interoperability across platforms and across versions of DFS. The bulk status RPC behaves gracefully across provider versions:

(a) When the server provides bulk status and the client does not, then bulk status is never invoked by the client.

(b) When the client provides bulk status and the server does not, then the server returns an out-of-range error when it fields a bulk status RPC. On its first bulk status interaction with a server that does not provide bulk status, the client gets this error and marks the server accordingly. The bulk status RPC is bypassed for this server thereafter.

It follows that, at steady state in the cell, clients only issue bulk status RPCs to servers that support bulk status. Otherwise the client falls through to lookup RPCs, short-circuiting the bulk status RPC.

4.5. Retry Policy

DFS RPCs that implement DFS semantics make exceptional efforts to complete successfully. Part of these efforts involve systematic retries on both the client and server sides of the RPC. The situation with bulk status is very different. As it does not contribute directly to DFS semantics, there is no imperative for the RPC to succeed. Because the bulk status RPC is an order of magnitude more heavy duty than other RPCs, the cost of retry efforts would be extraordinarily high. Unlike other DFS RPCs, there is a graceful fallback in case the bulk status RPC fails. Thus, the bulk status RPC is coded not to fall into the built-in retry paths of client-side error handling. There is no retry on either the client or the server.

4.6. Directory Handling

DFS RPCs that operate in a directory context normally verify that some other client machine, or even some other thread on the same client machine, has not changed the directory while the RPC was in-progress. The probability of this occurring during the bulk status RPC is much higher than it is for other RPCs because it runs over ten times longer. Thus the bulk status RPC makes explicit checks for other clients or other threads that might have changed the directory during the RPC and aborts if necessary, thereby falling through to conventional lookup behavior.

The first window of files for a directory is returned by the bulk status RPC with generalized status for the directory and its parent as the first two entries. The client discards this information as it is already in the name and status caches. This optimization handles the general case of smallish directories more efficiently.

5. THE SERVER ARCHITECTURE

A DFS server will in general be responding to many concurrent bulk status RPCs, so there is a concern to bound the resources required to support bulk status on the server. Accordingly, the bulk status RPC limits the amount of space it requires to marshal return parameters and selectively limits the number of tokens that it consumes.

5.1. Server Space Limit

There are several reasons why it is desirable to limit the space requirement on the server for a bulk status RPC instance:

(a) The bulk status RPC is long-running and this may influence the number of outstanding RPCs in progress at one time, hence influence the cumulative space requirement.

(b) The bulk status RPC is in the path of one of the most commonly invoked user commands, influencing the number of outstanding RPCs and the space requirement.

(c) Marshalling the returned status array will anyway require upwards of 10 kbytes, already an order of magnitude more than any other RPC.

Accordingly, the bulk status RPC acts to sharply upper bound the space required to marshal the conformant array of returned status. The mechanism is to limit the number of files handled per-call to `AFS_BULKMAX`. The size of the returned generalized status for a file is 236 bytes, so if `AFS_BULKMAX` = 32 (for example), then the total required per-RPC for status would be 7552 bytes.

5.2. Token Handling

A DFS file server is confined to issuing a certain maximum total number of tokens for files. Therefore, it is desirable to selectively limit the number of tokens returned by the bulk status RPC to the client. The default maximum is `afsBulkStatMaxTok`, however the system administrator may reset this value to a smaller (or larger) number at runtime. As soon as this maximum is exceeded during a bulk status call, the server stops getting (and returning) tokens for the rest of the files in the window but does return the remaining generalized status with no error and with the token cleared.

The bulk status RPC is somewhat speculative in that it is preloading generalized file status on the client with a prospect but not a promise of gain. In any case the bulk status RPC does not contribute directly to DFS semantics, so it need not succeed in returning current generalized status for every file in the window. Furthermore it would be counterproductive for the bulk status RPC to get in the way of DFS semantics executing on other clients. Lastly, revoking a token in general involves an additional RPC; it seems counterproductive for a long running RPC to raise a cascaded RPC in its code path. Hence the server that handles a bulk status call does not revoke any existing token while handling the call. It follows that, although it seeks to return the most optimistic token for each file in the directory, the bulk status RPC may return less powerful tokens for some files in the window, and may not always succeed in getting even a `READ_STATUS` token for the file.

While processing filenames from the directory window, the server tries to get a token for each file. It may be the case that the server will have to drop the directory token somewhere in the (really quite convoluted) code that is executed while trying to get a token for a file in the directory. While dropped, the token may be grabbed by some other RPC. The indication to the caller of the token code is only that the token was dropped; it is an inference that it may have been lost. When this happens to a conventional DFS RPC, it retries in a safe manner until it gets the desired file token while retaining the directory token. This may take awhile, so it would seem that retrying here is not desirable for the bulk status RPC. Accordingly, the server bails out of the bulk status RPC whenever it has to drop the directory token. At this point it is not feasible to trust the generalized status for any file in the window. The code marks the status of each file invalid, clears the token field and returns an error for each file in the window. The error value returned by the token code is also returned by the RPC, and the client is coded to not retry in this case.

5.3. Disabling the RPC

The bulk status RPC is much more heavy duty than any other RPC and it does not contribute directly to DFS semantics, so it is conceivable that a system administrator would prefer to disable bulk status service on a server. By default service is enabled, however the system administrator may disable service at runtime.

5.4. When Generalized Status is not Returned

When the server does not return generalized status for a particular file, different conventions are used to so indicate:

(a) An absent token is indicated by clearing the token via a call to `px_ClearTokenStruct()`.

(b) An absent status is indicated by setting it to `Invalid`.

(c) An absent fid is indicated by zero in the fid field.

While processing filenames from the directory window, the server tries to find a fid for each file. In the event this fails for a particular file, the server sets the error code in the corresponding entry of the conformant array of generalized status and continues to the next file in the window.

While processing filenames from the directory window, the server tries to obtain the current status of the file. In case this fails the server marks the status as bad, sets the error code for the file in the array of generalized status fields and continues to the next file in the window.

5.5. Returning Directory Info

Almost all DFS RPCs that pass a directory as an in-parameter, pass back the current status of the directory and a token for the directory as out-parameters. The bulk status RPC does this too, however in case the directory token was dropped as described in a previous section, then it is not safe to do this. Thus when the RPC returns an error, then the returned directory status and token is not merged with the current status (in fact they are marked invalid and cleared, respectively).

6. STATISTICS COLLECTION

The bulk status RPC makes provision to collect bulk status related statistics, en suite with existing statistics collection, on the client machines. (The extent of statistics collection on the server side is to just count the number of times each RPC is called and the total number of RPCs. So on the server only these numbers are available.) The statistics collected reflect the efficiency with which the bulk status RPC uses the name and status caches. For each of these caches the following counters are kept:

(a) The number (since reset) of times that a bulk status installed entry was retrieved from the cache.

(b) The number (since reset) of times that a bulk status installed entry was recycled without having been accessed at all.

(c) The number (since reset) of entries to the cache that were installed by the bulk status RPC.

The necessary counters for the name cache are collected in the (pre-existing) `nh_stats` structure in the midst of other counters.

```
struct nh_stats
{
    /* ... */
    int bulkStatSeen;
    int bulkStatNotseen;
    int bulkStatEntered;
    /* ... */
} nh_stats;
```

The `bulkStatSeen` counter is incremented in `nh_lookup()`, the `bulkStatNotseen` counter is incremented when the entry is recycled in `nh_enter()` and when it is purged in `purge()`, and the `bulkStatEntered` counter is incremented in `nh_enter()`.

The necessary counters for the status cache are collected in the (pre-existing) `cm_stats` structure in the midst of other counters.

```
struct cm_stats
{
    /* ... * /
    unsigned long statusCacheBulkstatSeen;
    unsigned long statusCacheBulkstatNotseen;
    unsigned long statusCacheBulkstatEntered;
    /* ... */
} cm_stats;
```

The `statusCacheBulkstatSeen` counter is incremented in `cm_FindSCache()`, the `statusCacheBulkstatNotseen` counter is incremented in `FlushSCache()`, and the `statusCacheBulkstatEntered` counter is incremented in `cm_BulkFetchStatus()`.

7. BENCHMARKING

The purpose of the bulk status RPC is to improve performance in files per second of commands that consume the status of all the files in a directory. Three benchmarks are available to help evaluate the effectiveness of the effort:

(a) `BIGDIR` — `ls -l directory` is performed in a flat directory; I construct directories whose size ranges between 1000 and 10000 files, incrementing in units of 1000 files.

(b) `DEEPDIR` — `ls -lR directory` is performed in a balanced directory tree; I construct a number of directory trees that contain between 1000 and 10000 files, approximately, incrementing in units of 1000 files.

(c) `SEARCHDIR` — `find . -name joe` is performed in the DEEPDIR directory trees, where the file does not exist.

Anecdotally, these benchmarks show an average 50% performance improvement for the bulk status RPC, as implemented according to the above description, over conventional lookup, with the differential at its best for smaller directories and closing for larger directories, as of this writing.

8. SYSADMIN RUNTIME CONTROLS

The bulk status RPC gives the system administrator active control (currently via kernel mode `adb`) at runtime over many bulk status related parameters. The parameters described all default to something reasonable, so there is normally no need to change them:

(a) On the server:

- (i) To monitor the number of bulk status RPCs examine the structure `afsStats`.
- (ii) To control the maximum size of a directory window revalue `afsMaxFilesBulkStat` to some number between 2 and `AFS_BULKMAX`. Do not set this value to be less than 2 or larger than `AFS_BULKMAX`. The current value of `AFS_BULKMAX` is 32.
- (iii) To control the maximum number of tokens returned by the bulk status RPC revalue `afsBulkStatMaxTok` to some non-negative number. The current default is 100.
- (iv) To disable the bulk status RPC on the server clear `afsBulkStatServerEnable`.

(b) On the client:

- (i) To disable the bulk status RPC from this client clear `cm_EnableBulkStat`.
- (ii) To monitor the interaction of the bulk status RPC with the name cache examine the structure `nh_stats`.
- (iii) To monitor the interaction of the bulk status RPC with the status cache examine the structure `cm_stats`.

9. LAST COMMENTS

There are a few technical considerations that are as yet imperfectly realized in the bulk status RPC, as currently implemented:

(a) It is desirable to support a client option to limit the size of the directory window. At present the maximum size of the directory window is determined, within limits, by the server and by the formal definition of the RPC.

(b) It is desirable to support a client option to limit the number of tokens returned. At present there is a server option to limit the number of tokens returned and the client may ask the server to return no tokens at all for files.

(c) It is desirable to support a server option to return no tokens (supported by setting `afsBulkStatMaxTok` to 0) and further support a server option to return status for files in the directory window precisely when a token is available for the file. At present the server revokes no token to acquire a token, however the token code is a complex coil and it is possible for the desired token to be available but for no generalized status to be returned at all.

10. ACKNOWLEDGEMENTS

The following persons made contributions, in the form of good advice, comments and insistent suggestions, to the bulk status RPC:

(a) Craig Everhart (Transarc).

(b) Bruce Leverett (Transarc).

(c) Lyle Seaman (Transarc).

(d) Carl Burnett (IBM).

(e) James Mostek (Transarc).

AUTHOR'S ADDRESS

Jason Gait
Transarc Corp.
707 Grant Street
Pittsburgh, PA 15219
USA

Internet email: gait@transarc.com
Telephone: +1-412-338-6933