Objectivity

301B East Evelyn Avenue
Mountain View, CA 94041
1-650-254-7100

-

MRD for Client side decompression.

Approval

Department	*Name*	*Date*
Development		
Marketing		
Sales		
Support		

Modification History

Version	*Date*	*Author*	*Comments*
1.0	1-May-02	Brian Clark	Initial Draft

Feature Requirements

Description of the Problem

SLAC have implemented a way for the AMS to decompress previously (manually) compressed database files. However this implementation has been seen to produce excessive CPU load on the AMS server processors.

Description of the requested features

SLAC have requested we implement client side decompression using the oofs protocol. See Appendix A for details.

Part of an optional feature or does it require another feature? If so, which one?

This is an additional feature to the default OOFS implementation.

How is this problem being solved now, and why isn't that acceptable?

Decompression on the server side is used, but this produces unacceptable processing load on the AMS server CPU.

Which languages must support this capability?

Kernel feature.

Which platforms must be supported?

All.

Do any competitors already have this capability?

TBD

Benefit Category:

Performance and Scalability. (Improved network throughput and better use of disk space)

Customers who require this capability:

- SLAC.

- TRW will need this and a rewritable version.

Revenue at risk or which could be won:
TBD.

When is this required?
ASAP

Review

Feature Sizing

Efforts	*Size*
Development	
QA	
Documentation	

Scheduled for Objy Release
Assigned Engineering Group

ADDITIONAL NOTES

Andrew Hanushevsky wrote:

```
> What's needed.
>
> 1) When the NM issues an open() request to the ams, the ams may
respond
> along with a status code of 0 (OOFS_OK) with a message of the form
of:
>
> !attn C=abcd R=bytes
>
> where C= indicates that the file is compressed and abcd indicates the
> algorithm used to compress the file. The R= is the region size (i.e.,
the
> number of bytes compressed per region), and a power of 2.
>
> 2) If the region size is equal to the page size, client-side
decompression
> is allowed. The NM then calls
>
>     class oocx_Compress *oocx_CX_Object(char *cxid)
>
>     where cxid points to the compression id (i.e., abcd).
oocx_CX_Object()
> either returns NULL, indicating that the compression mode is not
supported
> or returns a pointer to an oocx_Compress object. The object is
defined as
> follows:
>
> class oocx_Interface
> {
> public:
>
> virtual unsigned char *Compress(const unsigned char *data, long dlen,
>                                                     long &alen,
> unsigned char *buff=0) = 0;
> virtual unsigned char *Expand    (const unsigned char *data, long
dlen,
>                                                     long &alen,
> unsigned char *buff=0) = 0;
>
>            const       char *LastError() {return (const char
*)errbuff;}
>
>              oocx_Interface() {errbuff=0;}
> virtual ~oocx_Interface();
>
> protected:
>
> char *errbuff;
> };
>
> 3) When an object is returned, the file can be decompressed (i.e.,
expanded)
> on the client's side. At this point,
```

```
>     the NM can notify the AMS that it wants to read data in
compressed mode
> (i.e., the server is not to expand the data).
> This is done via the setOptions() (a member function of the oofs_File
class)
> with OOAMS_RAWIO as the argument. The definition is given below.
>
> #define OOAMS_RAWIO 0x00000001
>
> int setOptions(const unsigned long newopts, unsigned long &oldopts,
> oofsCredentials *credentials=0);
>
> 4) Should setOptions() not be called  the AMS server handles all
> decompression. Otherwise, the AMS server returns a compressed page to
the
> caller whose size will not be greater than the amount originally
requested
> and will always represent a full page after decompression. In fact,
if the
> number of returned bytes is equal to the page size, the Expand()
method
> should not be called because the page is not compressed. Otherwise,
Expand()
> is called to decompress the page.
>
> 5) The result (decompressed or not) is returned to the upper layers.
>
> -------------------------------------
>
> Some issues:
>
> 1) The easiest way to include compression support is to link in a
dummy
> liboocx.so that contains a get_oocx_Object() routine that always
returns
> NULL. This would then make the whole thing transparent. If one does
not want
> to distribute a dummy liboocx.so then the NM must look for the
library at
> initialization time and use dlopen() and dlsym() to manually load the
> routine to be used at runtime, should the library be found. It would
appear
> that the "dummy" library approach is probably the easiest to
accomplish.
>
> 2) The setOptions() method is new and would represent an addition to
the
> outgoing protocol as well as a change in the oofs interface. While
these are
> not necessarily bad, they could potentially be disruptive to certain
> segments of the community (certainly not SLAC or it's collaborative
> members). There is really only one solution to this problem:
>
>     a) abscond with an unused open flag or redefine an existing flag
(e.g.,
> O_NDELAY could be used to indicate that raw I/O is wanted -- like
don't
```

> delay decompressing the page). In either case, care should be taken
to make
> sure the NM is not setting the bits randomly or otherwise passing
them
> trough when not wanted). Of the two options, reusing an existing flag
is by
> far the safer choice.
>
>   b) Should the NM find that it cannot handle decompression for a
> particular file (i.e., either the region size is not equal to the
page size
> or the algorithm is not supported) it would have to close the file
and
> reopen it with the "special" flag turned off. This will likely not
happen
> too often and, perhaps, merits a warning message from the NM when it
does
> happen.
>
> All in all, for the least disruptive change, the inelegant open/
close/open
> method in handling decompression makes the most sense.
>
> 3) Configuring the system for client-side decompression is really a
> no-brainer. In general, client-size decompression is always enabled.
It can
> be selective disabled on the server side, as needed. If the client
really
> wants to disable client-side decompression (something we would prefer
not to
> happen easily), then simply putting in a "dummy" liboocx.so (or
removing it
> altogether for manual load operations) would be sufficient for those
cases
> where a client does not want client-side decompression. Another
possible
> approach would be setting an environmental variable (i.e.,
OONM_NOINFLATE)
> would also disable client-side decompression.
>
> 4) My estimate of the changes is about 100 lines of code, most of
which
> would be used to parse the message response from the server. We would
> happily provide the class definition and the dummy library since
these
> really don't represent any intellectual property (they don't do
anything).
>
> Let us know whether you can schedule this in.
>
> Andy