Market Requirements Document

Feature Name: Enhanced Client-Side Caching

Version: 3	**Date Submitted:** 11/04/08	**Completed By:** Leon Guzenda
Version: 2	Date Submitted: 11/04/08	Completed By: Leon Guzenda
Version: 1	Date Submitted: 10/28/08	Completed By: Leon Guzenda

Description of the Problem

Background

Objectivity/DB currently uses a Least Recently Used discard algorithm to move pages out of the client process/thread cache when more space is required to create new objects or to read existing pages from disk.

Abbreviations

- LFU – Least Frequently Used
- LRU – Least Recently Used
- MFU – Most Frequently Used
- MRU – Most Recently Used

The Problem

The current algorithm works well if data is being streamed to or from disk, i.e. a page is changed and can then be written to disk and discarded, or a page is read, used and can then be discarded. It also works fairly for random access applications that only use pages once in a transaction, or infrequently over the life of the cache. However, it is not very good at supporting applications that stream data and correlate it with a set of existing data.

Consider an application that is reading an incoming stream of telephone call detail records and maintaining a count of the number of callers and callees per central office (Area Code + next four digits). The number of central offices is fixed and it is useful to cache all of them in memory to avoid I/Os. However, as the stream of call details records is read into memory the cached pages containing central office details will be paged out, even though there is a high likelihood that they will be needed again.

The only current solution to this problem is for the application to explicitly hold the call detail objects open (or to pin them in RAM).

Description of the Requested Feature

The caching algorithm should be able to:

1. Manage objects that are smaller than a page.
2. Manage objects that are larger than a page.
3. Allow holding or pinning of objects within RAM.
4. Allow the user to define the initial size, growth rate and maximum size of cache areas according to the types of data and usage (small objects, large objects/ LVArrays, indices, collections, schema information, file housekeeping, iteration/ streaming etc.). See the Notes section for qualifications of this requirement.
5. Favor pages that are more frequently used than others.
6. Discard equally used pages in LRU order.

Part of an existing feature or does it require another feature, if so, which one?

This would be a standard product feature.

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

1. Users have to explicitly pin objects in the cache. This is less efficient than being able to control the page discard mechanism.
2. LVArrays have to fit in memory, so large streams of data have to be artificial broken into logical/physical pages. There are separate MRDs covering chunked/ streamed LVArrays and advanced array handling.

What languages must support this capability?

- C++, Python, Java and .Net for C# in that order of priority.
- Hopefully, any kernel change will quickly propagate to other APIs.

Which platforms must be supported?

- Linux, Windows and Unix.

Do any competitors already have this feature?

- No, the closest is ObjectStore, which allows a user to swap whole partitions of files into and out of RAM. This can allow a transaction to start by preloading information, switch to streaming and accumulating transient information, then switch to the original information and update it before completing the transaction.

- Some products support direct streaming.

Customers who require this feature

- Scientific, engineering and data fusion users – These algorithms frequently access lookup tables while handling streams of data
- Complex financial simulations.
- Image processing and management applications.
- Medical equipment applications, e.g. for CAT scans and experiment results.
- Customers building data mining tools for very large datasets.

Revenue at risk, or which could be won

- There is no immediate risk, but we will be at a severe competitive disadvantage in the scientific, engineering, energy and complex financial markets once details of the SciDB API become public, which could be as soon as early-mid 2009.

When is this required?

- Release 10 (Performance), but see the Notes section below for qualification.

Additional Notes

We will also need:

- Marketing collateral, including promotional material and a special area on our web site.

- Technical Publications and updates to conventional and web based training material..

- New QA material to prove that the mechanism works, is effective and is interoperable with other platform and language combinations.

If an MFU algorithm is introduced it may be necessary to allow the user to reset the counts to 1 to avoid favoring pages that are heavily used in a single transaction and then never accessed again. Alternatively, the counts could be reduced by some proportion (e.g. 25%) at the end of each transaction.

The cache control parameters set by the user need not be extended for Release 10, e.g. there is no need to implement streaming behavior for LVArrays, nor to allow control over schema,index or other housekeeping page areas.