Apache DirectMemory - initial roadmap discussion

<Please notice: this is a repost of the first email sent to the ML - as not everyone has seen it I post it as a blog entry to keep track of the discussion>

Gentlemen, welcome and thank you for joining in (and the opportunity, for me and the project, to join the ASF, which is great) . I wrote some notes about the current state of the project and some hypothesis on future developments which I would like to discuss with you all. These are the items I would like to discuss (and sorry for being a bit lengthy):

- **Design choices**
- **New features**
- **Integration** with other products
- **Build, Test and Continuous integration** strategy
- **Miscellanea**

Design choices

I recently rewrote DM entirely for simplification. It used to have three layers (heap, off-heap, file/nosql) and to authomatically push forward/backward in the chain items according to their usage. It turned out overly complicated and mostly inefficent at runtime (probably mostly because of my poor implementation). The singleton facade is proving simple and effective and well refects the nature of direct memory - which cannot be really freed. But this needs a strategy for feature and behaviour composability.

- **Singleton** (largely Play! inspired) approach - is it good?
- **Feature and behaviour composability** (DI and Feature injection? A plugin system? OSGi?) - just let's keep things simple and developer friendly

New features

Adding simple heap cache features would spread usage among those who think that would EVENTUALLY need a huge off-heap one (I believe it's the vast majority of our potential "customers"). Same thing for file and distributed ones. Having both three would qualify DM as an Enterprise Ready (please notice the capitalization 😁 cache.

- **Heap storage** - Guava already fits the requirement, of course. We could both use the heap as a "queue" to speed up inserts and serialize later and/or keep most frequently used items into the heap for speed. It's more a design choice than a technical one
- **File storage** - this would be easy to achieve with the same "index" strategy of the off-heap one (I believe JCS does the same)
- **Lateral storage** (distributed or replicated) - A possible way to do this: hazelcast for map distribution and Apache Thrift for intra node communication (node a needs an item stored in node b and then asks for it). I'm not sure hazelcast would perform as well as Guava with multi million item maps, it has to be thoroughly tested for perfomance and memory consumption - should hazelcast not fit the performance requirement we should finda an alternative way to distribute/replicate the map across nodes. jgroups with multicasting would be perfect but it's LGPL (well, JCS uses it) and, of course, a custom, maybe thrift based, distribution mechanism could be written ad-hoc

Integration with other products

Providing plugins, integration or just support with/for other technologies/products would of course spread adoption. These are the first few that pop in my mind at the moment

- **Apache Cayenne** integration - as a second level query cache
- **Play! Framework** integration - because I simply love play! and use it in other side projects whenever I need a web/mobile fron-end
- **Memcached (like)** integration - DirectMemory can be seen as an embedded memcached and adoption its protocol would be a good fit for replacing it in distributed scenarios most of all when it's used by java applications
- **Scala, Clojure and other jvm languages** integration - emerging technologies that deserve attention. Should I have 48 hours days I would use the other 24 to improve my scala skills and rewrite DirectMemory can with it

Miscellanea

There are of course a lot of things that are not essential but could be investigated

- **HugeArrayList, FastMap**, etc... DirectMemory currently uses Guava for the Map and ArrayList (I know it's not thread safe but it could be really not required) for the Pointer's index. Evaluation of other fast and low memory impact Map and List implementation could possibly bring performance improvements
- **Reliability improvements** - DirectMemory is fast also because it sacrifices reliability - is it always a good trade-off? Could we provide configuration or pluggable implementation for different usage scenarios, maybe at list for the MemoryManager? Or even transactionality?
- **Would hadoop need off-heap caching**? (this is a good one)

Build, Test and Continuous integration strategy

The overall point for DM is testing for performance with large quantities of memory - where the minimum should be more than the average 2GB used by web applications - the more the better.

- **Testing infrastructure** - I currently use an amazon machine with 16+GB RAM (which costs ~$1 per hour), a bit tedious and time consuming to startup and to deploy on (would require some scripting) and of course continuous performance testing is too expensive - alternatives?

- **Branching strategy** - I don't like feature branches - I believe feature composability should not be done at the SCM level - (and SVN is probably a bit too slow for them) and don't believe in using just release branches. Don't know whether there's an apache standard but I usually work with spike branches (where a spike is more than a single feature and less than a whole release) and then publish on release branches tagging for events (production, distribution, etc). Does it sound good for you?
- **Binary packaging and demo applications** - I used to provide a binary distribution and a simple web application to test against but it simply was too effort for me alone
- **OSGi bundling** - it costs very little and can be quite useful
- **Maven repository** - I've applied for a sonatype repo registration but simply didn't have enough time to complete it and I'm using a github folder as a repository. I guess that artifacts would naturally go in apache repos, from now on, right?
- **Testing and certification** over different JVMs and OS (sun, openjdk, ibm, windows, linux, AIX? Solaris?)

Roadmap

I would say that intensive performance testing and certification would make a solid 0.7 GA release; heap and file storages inclusion would make a pretty good 1.0 (the distributed storage would make it incredible!)

Waiting forward for your feedback.

Cheers,
Raffaele