14/9/2013 VIRGO Malloc
Design notes - Contd

Virgo_malloc() design option I that wraps calls to multiple ~~[illegible]~~ malloc() calls and concatenates the malloc()-ed memory chunks spread across multitude of cloud nodes, is quite a high-level simple implementation that delegates the complex issues of memory allocation, alignment etc., to the OS level code. Also virgo_malloc is quite useful in allocating large kernel memory virtualized and scattered across cloud nodes. Thus all individual malloc commands sent to the remote virgo_cloudexec service are kernel malloc calls. Thus virgo_malloc() allocates very large kernel memory on the cloud & not userspace memory (if in kernel mode

execution). For userspace virgo_malloc() virgo_cloudexec() makes upcall and does malloc() in userspace and returns it to the remote virgo_malloc(). This is somewhat circuitous since this userspace allocation can be done in userspace itself without going to kernel. Thus

services in cloudnodes accept malloc requests and return allocated chunks using only userspace sockets, without any kernel sockets. Virgo_malloc() is thus useful when large kernel memory is needed sprawling across

JULY 2013	M	T	W	T	F	S	S	M	T	W	T	F	S	S
								1	2	3	4	5	6	7
	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	22	23	24	25	26	27	28	29	30	31				

few thousand cloud nodes. Internally kmalloc() is invoked.

But when large kernel memory is spread across thousands of cloud nodes, it is still left to be researched. Probably this would be useful in viewing individual OS instances in cloud nodes together as a SINGLE KERNEL MONOLITH.

Ka Shrinivaasan
14/9/2013

Implementation algorithm:

1) Replicate VIRGO clone() code base and change the code for virgo malloc()

2) VIRGO address translation table for VIRGO virtual addressing.

3) Testcases.

Ka Shrinivaasan
14/9/2013

SEP 2013	M	T	W	T	F	S	S	M	T	W	T	F	S	S
								1	2	3	4	5	6	7	8
	9	10	11	12	13	14	15	16	17	18	19	20	21	22
	23	24	25	26	27	28	29	30						