## NAME

jumbomem − enable programs to page to remote RAM instead of to local disk

## SYNOPSIS

**jumbomem** [−−**help**] [−−**version**] [−−**nodes**=*count*] [−−**debug**=*level*] [−−**pagesize**=*bytes*] [−−**heartbeat**=*seconds*] [−−**reserve**=*bytes*|*percent*%] [−−**slavemem**=*bytes*] [−−**mastermem**=*bytes*] [−−**pages**=*count*|*percent*%] [−−**rankvar**=*variable*] [−−**baseaddr**=*address*|+*bytes*] [−−**prefetch**[=**none**|**next**|**delta**] [−−**fast−start**] [−−**async−evict**] [−−**memcopy**] [−−**nre−entries**=*count*] [−−**nre−retries**=*count*] [−−**nru−interval**=*milliseconds*] [−−**true−nru**] [−−**mlock**] *command*

## DESCRIPTION

JumboMem provides a low-effort solution to the problem of running memory-hungry programs on memory-starved computers. The JumboMem middleware gives programs access to all of the memory in an entire cluster, providing the illusion that all of the memory resides within a single computer. When a program exceeds the memory in one computer, it automatically spills over into the memory of the next computer. Behind the scenes, JumboMem handles all of the network communication required to make this work; the user's program does not need to be modified — not even recompiled — to take advantage of JumboMem. Furthermore, JumboMem does not need administrator privileges to install. Any ordinary user with an account on a workstation cluster has sufficient privileges to install and run JumboMem.

## OPTIONS

**jumbomem** accepts the following command-line options (in roughly descreasing order of usefulness):

−−**help** Output a brief usage summary.

−−**version**
Display the curent version of JumboMem.

−**np** *count*, −−**nodes**=*count*
Specify the number of nodes to use. *count* should be at least `3`: one master plus two slaves. The more nodes you have available, the more memory you can use.

−−**debug**=*level*
Control the amount of debugging information that JumboMem outputs (default level: `1`). See the NOTES section below for a detailed description of what gets output at each debug level.

−−**pagesize**=*bytes*
Designate a logical page size for JumboMem to use. The default is `262144` (256KB). Applications with a high degree of spatial locality (i.e., those with largely contiguous data accesses) generally perform better with large pages. Applications with a low degree of spatial locality (i.e., those with essentially random data accesses) generally perform better with small pages.

−−**heartbeat**=*seconds*
At debug levels 1 and up, output a status message every *seconds* seconds indicating the number of major page faults observed by the operating system (should be close to 0) and by JumboMem (can be low or high, depending on the application).

−−**reserve**=*bytes*|*percent*%
Reserve either *bytes* bytes or *percent*% of the available memory for use by the operating system or other, non-JumboMem processes. The default is `1%`. This number should be increased if a JumboMem process induces a non-negligible number of major OS page faults as these can incur a significant performance penalty.

−−**slavemem**=*bytes*
Specify explicitly the amount of memory that each slave process can serve. While −−**reserve** specifies how much memory JumboMem should not use, −−**slavemem** instead specifies how much memory JumboMem should use.

−−**mastermem**=*bytes*
Specify explicitly the amount of memory that the JumboMem master process is allowed to use for its local cache of remote pages (and miscellaneous data structures). While −−**reserve** specifies how much memory JumboMem should not use, −−**mastermem** instead specifies how much

memory JumboMem should use.

**−−pages**=*count*|*percent*%
: Limit the number of logical pages that JumboMem is allowed to cache locally to either *count* or to *percent*% of the number of page mappings supported by the operating system. The default is `70%`. The intention is to further avoid local paging to disk, as this severely degrades the performance of a process run with JumboMem.

**−−rankvar**=*variable*
: Designate an environment variable that distinguishes the master process (rank 0 in the computation) from the slave processes (all other ranks). The **findrankvars** MPI program included with JumboMem suggests variables to use with **−−rankvar**.

**−−baseaddr**=*address*|**+***bytes*
: Force JumboMem to allocate its memory region at memory location *address* or at *bytes* bytes past the default location (the end of the program's data segment rounded up to the nearest JumboMem page size). JumboMem aborts if it cannot allocate its memory region at the specified address or address delta. Note that JumboMem will ensure that its memory region begins on a multiple of the JumboMem page size, rounding up *address* (or *default*+*bytes*) if necessary.

**−−prefetch**[=**none**|**next**|**delta**
: Enable prefetching of remote pages. Most empirical tests of JumboMem indicate that prefetching in fact degrades performance so the default is `none`: no prefetching. However, on some networks or MPI implementations enabling prefetching may improve performance. Specifying **−−prefetch**=**next** causes the statically subsequent page to be prefetched on every page access. Specifying **−−prefetch**=**delta** or just **−−prefetch** induces prefetching of the page at the same distance from the previous fetch. For example, after fetching pages *i* and *i*+3 JumboMem would prefetch page *i*+6.

**−−fast−start**
: Prevent JumboMem's initial calibration of reasonable memory sizes. Normally, as part of JumboMem initialization, each processs allocates a large region of memory (the size it expects to be able to use), accesses all of the data in it, and, if any major OS page faults were detected, reduces the size of the region and tries again. Doing so helps reduce the number of major OS page faults. **−−fast−start** tells JumboMem to skip this step, thereby initializing faster but possibly running slower. **−−fast−start** is also good for benchmarking because it ensures that the same memory sizes are used across runs.

**−−async−evict**
: Evict pages without waiting for the eviction to complete. Most empirical tests of JumboMem indicate that this does not improve performance so the default is to wait for evictions to complete. However, on some networks or MPI implementations enabling asynchronous evictions may improve performance.

**−−memcopy**
: Copy pages into a static communication buffer before transmitting them and copy pages from a static communication buffer after receiving them. On most networks and MPI implementations these extra copies degrade performance. However, on some connection-based networks, limiting the number of registered (a.k.a. pinned) memory regions may compensate for the extra copies in terms of performance.

**−−nre−entries**=*count*
: When using NRE (not recently evicted) page replacement, keep track of the *count* most recently evicted pages. If a (randomly selected) victim page is one of the *count* most recently evicted pages, the algorithm selects a different victim page.

**−−nre−retries**=*count*
: When using NRE (not recently evicted) page replacement, if a victim page was recently evicted, the algorithm selects a different victim page. This process repeats *count* times before the algorithm gives up and evicts a recently evicted page.

**--nru-interval**=*milliseconds*

When using pseudo-NRU (not recently used) page replacement (the default when using the NRU page-replacement module), clear all ''accessed'' bits every *milliseconds* milliseconds. NRU replacement schemes favor replacing pages that have not recently been accessed over pages that have. The default is `5000` (5 seconds).

**--true-nru**

Use a true NRU (not recently used) page-replacement scheme instead of the default pseudo-NRU scheme when running with the NRU page-replacement module. A true NRU scheme distinguishes between dirty (modified) and clean (unmodified) pages. Clean pages are favored for replacement because evictions do not require network traffic. Unfortunately, because JumboMem is an entirely user-level system, keeping track of each page's modification state requires a significant amount of extra work per fault, and this typically results in worse performance than simply assuming that all pages are dirty. Applications that performance significantly more reads than writes may benefit from **--true-nru**.

**--mlock**

Attempt to use `mlock()` to lock memory pages into RAM while in use. Use of this option may improve performance by reducing the number of major OS page faults. However, it may also lead the OS to deem the JumboMem master or slaves to be ill-behaved processes and therefore subject to spontaneous termination by the OS.

A command to run follows the **jumbomem** options. This can be any sequential program, subject to the restrictions listed under RESTRICTIONS below.

Options that accept a number of bytes accept the following suffixes to the *bytes* argument:

**k** (kilobytes)
Multiply *bytes* by 1024.

**m** (megabytes)
Multiply *bytes* by 1,048,576 ($1024^2$).

**g** (gigabytes)
Multiply *bytes* by 1,073,741,824 ($1024^3$).

If a suffix is not specified, the *bytes* argument can be specified in either decimal, octal, or hexadecimal using ordinary C notation: Numbers beginning with `0x` are treated as hexadecimal; numbers beginning with `0` are treated as octal; and all other numbers are treated as decimal.

## EXAMPLE

Here's how to run an interactive Python session with access to 15 machines' worth of available RAM (55GB on the cluster on which this was tested):

```
jumbomem -np 16 python -i
```

Note that we specify `-np 16` because we always need one extra machine to serve as the master, which actually runs the application; the other machines in the cluster function as memory servers. The Python interpreter's `-i` option forces an interactive session. Without it, Python may conclude that it's running in noninteractive mode (because it was launched from some job-launching daemon such as **mpirun**'s) and not display a prompt, although it will otherwise work as normal.

## ENVIRONMENT

Most of the options to the **jumbomem** script merely set environment variables that the JumboMem runtime library (*libjumbomem.so*) reads and processes. The following environment variables are currently recognized:

JM_ASYNCEVICT

Corresponds to the **--async-evict** option when set to `1`; to the default case when set to `0`.

JM_BASEADDR

Corresponds to the **--baseaddr** option.

JM_DEBUG
: Corresponds to the `--`**debug** option.

JM_HEARTBEAT
: Corresponds to the `--`**heartbeat** option.

JM_LOCAL_PAGES
: Corresponds to the `--`**pages** option.

JM_MASTERMEM
: Corresponds to the `--`**mastermem** option.

JM_MEMCPY
: Corresponds to the `--`**memcopy** option when set to `1`; to the default case when set to `0`.

JM_MLOCK
: Corresponds to the `--`**mlock** option.

JM_NRE_ENTRIES
: Corresponds to the `--`**nre-entries** option.

JM_NRE_RETRIES
: Corresponds to the `--`**nre-retries** option.

JM_NRU_INTERVAL
: Corresponds to the `--`**nru-interval** option.

JM_NRU_RW
: Corresponds to the `--`**true-nru** option when set to `0`; to the default case when set to `1`.

JM_PAGESIZE
: Corresponds to the `--`**pagesize** option.

JM_PREFETCH
: Corresponds to the `--`**prefetch** option.

JM_RANKVAR
: Corresponds to the `--`**rankvar** option.

JM_REDUCEMEM
: Corresponds to the `--`**fast-start** option when set to `0`; to the default case when set to `1`.

JM_RESERVEMEM
: Corresponds to the `--`**reserve** option.

JM_SLAVEMEM
: Corresponds to the `--`**slavemem** option.

Note that unlike the corresponding command-line options, environment variables that specify a number of bytes do not accept a **k**, **m**, or **g** suffix.

## FILES

*libjumbomem.so*
: The core part of JumboMem. **jumbomem** loads *libjumbomem.so* into a process's memory using the dynamic linker's `LD_PRELOAD` envionrment variable. *libjumbomem.so* then installs its own memory-allocation and fault-handling routines.

*$HOME/.jumbomemrc*
: A configuration file used by **jumbomem**. After setting default values for various environment variables, **jumbomem** executes the user's *$HOME/.jumbomemrc* file, which generally contains code to set environment variables (see ENVIRONMENT above) but can in fact include arbitrary Bourne-shell code. For example, putting `JM_DEBUG=2` in your *$HOME/.jumbomemrc* changes the default debug level to 2. Note that **jumbomem** automatically exports all shell variables whose name begins with `JM_` so an explicit `export` statement is unnecessary.

*/proc/sys/vm/max_map_count*

On Linux, this ASCII file specifies the maximum number of memory-mapped regions a process can allocate. It typically defaults to 65,536. JumboMem's minimum logical page size is calculated as the total address-space size divided by the maximum number of memory-mapped regions. Hence, a larger number stored in */proc/sys/vm/max_map_count* implies greater flexibility in selecting a JumboMem logical page size. System administrators may therefore want to write a large number to */proc/sys/vm/max_map_count* on each node in their cluster to improve performance on applications that exhibit poor spatial locality. (See the description of **−−pagesize** in the OPTIONS section.)

## CAVEATS

Buggy programs that may have happened to work with smaller amounts of memory will likely fail when run with large amounts of JumboMem memory. For example, programs that use 32−bit integers to keep track of the number of elements in a data structure will fail when run with $2^{32}$ (or even $2^{31}$) elements; programs that are sloppy with pointer arithmetic (e.g., copying a 64−bit `void *` into a 32−bit `int` and back again) will fail when those pointers pass the 32−bit boundary. Check your programs before trying to run them with JumboMem!

## BUGS

Probably plenty. Please report any you can reproduce to the author. (See ''AUTHOR'' below.)

## RESTRICTIONS

While JumboMem works well for a number of programs it does not work well for all programs:

- JumboMem cannot handle processes that `fork()` other processes. This is probably the biggest limitation on JumboMem's utility.
- Only programs that use 64−bit pointers can take advantage of the extra memory provided by JumboMem.
- JumboMem manages only memory dynamically allocated with `malloc()` and related functions. Statically declared data structures are not distributed across the network.

## NOTES

### Troubleshooting

As a user-level program, JumboMem has to rely on a substantial amount of trickery to convince the target program that vast amounts of memory are available to it. Unfortunately, different programs react differently to all this trickery, which can lead to crashes, **jumbomem** error messages (e.g., `mmap() failed to allocate` *`number`* `bytes at or above address` *`address`*), or other problems.

Sometimes it helps to to experiment with using different base addresses for JumboMem's global address space. (See **−−baseaddr** in the OPTIONS section above.) The default is to start the global address space right after the program's data segment. This has the advantage of serving as a crutch for buggy programs by enabling as much data as possible to lie beneath the 32−bit boundary. The disadvantage is that some programs expect to be able to manage all of the data up to that boundary and consequently end up fighting for control with JumboMem. Try **−−baseaddr**=4g and see if that gets the problematic program to work.

A common configuration mistake is to omit or incorrectly specify the LAUNCHCMD command such that **jumbomem** launches all slave processes on the same node, exhausting that node's memory. If **jumbomem** appears to induce excessive disk paging or unexpectedly runs out of memory, it may be worth scrutinizing the LAUNCHCMD variable in *custom.py*. For quick testing you can edit the `launchtemplate` line in the *jumbomem* script; edit *custom.py* and rebuild JumboMem when a suitable LAUNCHCMD is found.

Other problems with JumboMem may be deciphered by increasing **jumbomem**'s debug level. See **−−debug** in the OPTIONS section above and Debug Levels below.

### Debug levels

At debug level 0, **jumbomem** outputs no additional information. At level 1 (the default), **jumbomem** outputs at the start of the run an ‘‘initializing’’ message and the values of some key JumboMem environment variables (including those set by the **jumbomem** script). At the end of the run **jumbomem** outputs an ‘‘exiting’’ message. Level 2 increases **jumbomem**’s verbosity to include a large amount of configuration detail during initialization and fault statistics during finalization. At level 3, each slave process additionally outputs configuration information and termination statistics. In addition, **jumbomem** announces every major increase in the amount of memory managed by **jumbomem** (i.e., every invocation of `morecore()`). **jumbomem** output becomes very verbose at level 4, at which point every page fetch, eviction, and permission change and every thread freeze/thaw request is output. Finally, at debug levels 5 and up, **jumbomem** outputs every entry to and exit from a memory-allocation function such as `malloc()` and `free()` and every thread freeze/thaw response.

### Heartbeat output

The JumboMem heartbeat value (set by **--heartbeat** or JM_HEARTBEAT) is checked only on JumboMem page faults. Consequently, an application that rarely page faults will not regularly output the heartbeat status message.

## AUTHOR

Scott Pakin, *pakin@lanl.gov*

## COPYRIGHT AND LICENSE