A Concurrent Real-Time White Paper

2881 Gateway Drive
Pompano Beach, FL 33069
(800) 666-4544
real-time.ccur.com

Memory-Shielded RAM Disk Support in RedHawk™ Linux® 6.0

By: John Blackwood
Concurrent Real-Time
Linux® Development Team
March 23, 2011

Abstract

This white paper describes a feature of RedHawk Linux that provides the ability to control NUMA node page placement of RAM disk data. Placing RAM disk pages in the same NUMA node where the RAM disk applications execute can result in higher performance and reduced system-wide memory contention. The paper describes several test case scenarios that demonstrate the benefits of proper RAM disk page placement.

Overview

In Non-Uniform Memory Architecture (NUMA) computer systems, memory access time depends upon the memory location relative to a processor. NUMA multiprocessor systems are made up of two or more NUMA nodes, where each node contains one or more processors and memory. In NUMA systems, all of memory is accessible to all processors. Memory accesses can be either local or remote -- local accesses to memory that resides on the same node as the processor, and remote accesses to memory that resides in a different node than where the processor resides. Remote accesses can be more expensive in terms of latency and increased bus contention, especially where the remote memory resides several NUMA node bus interconnections away from the accessing processor.

Historically, RedHawk Linux has provided the ability to memory-shield a NUMA node. Only user-space pages belonging to applications that are biased to execute on the memory-shielded NUMA node will reside in that node's memory. Memory-shielding also moves user-space pages of tasks that are executing on other NUMA nodes out of the memory-shielded node's memory. RedHawk memory-shielding support greatly increases the amount of local memory accesses that a memory-shielded application makes and reduces the amount of cross-node memory bus contention.

A RAM disk is a portion of RAM/memory that is used as if it were a disk drive. RAM disks have fixed sizes and act like regular disk partitions. Access time is much faster for a RAM disk compared to a physical disk. Data stored on a RAM disk is lost when the system is powered down; therefore RAM disks provide a good method for storing temporary data that require fast access.

In RedHawk 6.0, support has been added that gives the user the ability to specify the NUMA node(s) where the pages of a RAM disk will reside. A user may control, on a per-RAM disk basis, the NUMA nodes that will hold the memory data of the RAM disk. This new feature can further enhance the performance of memory-shield applications.

RAM Disk NUMA Node Interface

A user can specify the NUMA node page placement of a RAM disk through the use of */proc* file interface. Additionally, the per-node page count values for a RAM disk may also be viewed.

The */proc/driver/ram/ram[n] /nodemask* file may be used to view and to set the NUMA nodes where the RAM disk pages should reside, where the nodemask is a hexadecimal bitmask value. By default, RAM disk pages may reside on any NUMA node. For example, the default nodemask for RAM disk 2 on an 8 NUMA node system would be:

```
# cat /proc/driver/ram/ram2/nodemask
00000000,000000ff
```

In order to setup RAM disk 2 so that its pages are allocated in NUMA node 3, the following commands are issued to set and check the nodemask value:

```
# /bin/echo 8 > /proc/driver/ram/ram2/nodemask
# cat /proc/driver/ram/ram2/nodemask
00000000,00000008
```

After RAM disk pages have been allocated for a RAM disk, the */proc/driver/ram/ram[n]/pagestat* file may be used to view the per-node page counts. For example:

```
# cat /proc/driver/ram/ram2/pagestat
Preferred  node  3 pages:   2048
```

Performance Measurements

This paper discusses two test scenarios that demonstrate the potential performance and determinism advantages of using RAM disk pages that are local to the processes accessing the RAM disk compared to RAM disk pages that reside in a remote NUMA node. The RedHawk Linux memory-shielded RAM disk support was used in these tests to control the NUMA node placement of the RAM disk pages.

Although the test scenarios are not necessarily typical of the type of memory and RAM disk usage patterns in actual application, some portion of the differences observed in these test scenarios would be seen in actual real-world applications. The tests were executed on a quad-processor Opteron™ system with four NUMA nodes and four cores per node. The tests described below utilized two of the four NUMA nodes in the system.

Testing Methodology

A file write/read program was used in both of the two test scenarios. This program opens either a */dev/ram[n]* file directly, or opens a file that resides within the already mounted RAM disk file system. The *O_DIRECT open*(2) flag is used so that the RAM disk accesses bypass the file system page cache and cause direct file data copies between the user's buffers in user-space and the RAM disk pages in kernel-space.

The program first obtains a starting time using *clock_gettime*(2), and then executes either a loop of *lseek*(2) and *write*(2) calls, or a loop of *lseek*(2) and *read*(2) calls on the open file. After executing this set of system service calls, *clock_gettime*(2) is again used to obtain the ending time.

The pseudo-code for the lseek and write loop is shown below:

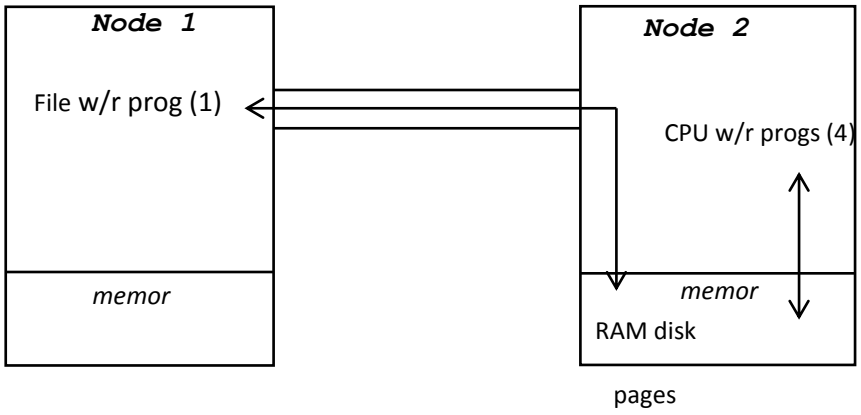
```
clock_gettime(CLOCK_MONOTONIC, start);
for (loop = 0; loop < loopcount; loop++) {
        lseek(fd, 0, SEEK_SET);
        write(fd, buffer, size);
}
clock_gettime(CLOCK_MONOTONIC, end);
```

In addition to this RAM disk write/read loop test, a CPU write/read memory test that continually writes and reads to/from user-space memory buffers was used in the first test scenario below in order to generate CPU memory access traffic. This test was written in a way where these CPU memory accesses tend to cause CPU cache misses, thus resulting in more actual physical memory (non-cached) accesses.

Test Scenario 1

In this scenario, one file write/read program was executed on the first NUMA node, where the program opened */dev/ram0* and issued 8 MB writes and reads directly to the raw RAM disk device.

Using Remote RAM Disk Pages



Initially, the RAM disk pages were setup to reside on a second (remote) NUMA node.

With no other activity on the remote node, the timings for accessing the RAM disk were:

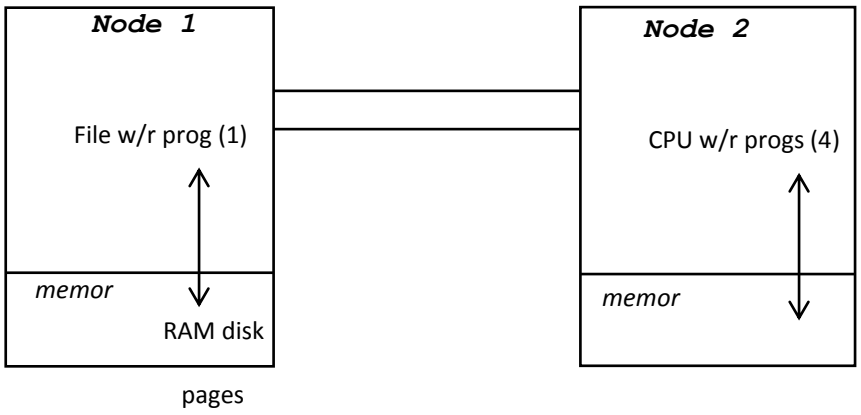
each lseek and write call: 8.916 milliseconds
each lseek and read call: 8.700 milliseconds

Then when four copies (1 copy per CPU) of the CPU write/read memory program were executed on the remote NUMA node where the RAM disk pages resided, the timings for accessing the RAM disk pages increased to:

each lseek and write call: 13.647 milliseconds
each lseek and read call: 12.981 milliseconds

Thus memory contention for accessing the remote RAM disk pages caused an increase of 53% for write calls and 49% for read calls in this test scenario.

Using local RAM Disk Pages



The RAM disk pages were then placed in the local NUMA node.

The same single file write/read program was executed on the first NUMA node. When there was no activity on the second NUMA node, the timings were:

each lseek and write call: 8.414 milliseconds
each lseek and read call: 8.501 milliseconds

Then when the same four copies of the CPU read/write memory program were executed on the second NUMA node, the timings increased to:

each lseek and write call: 8.800 milliseconds
each lseek and read call: 8.917 milliseconds

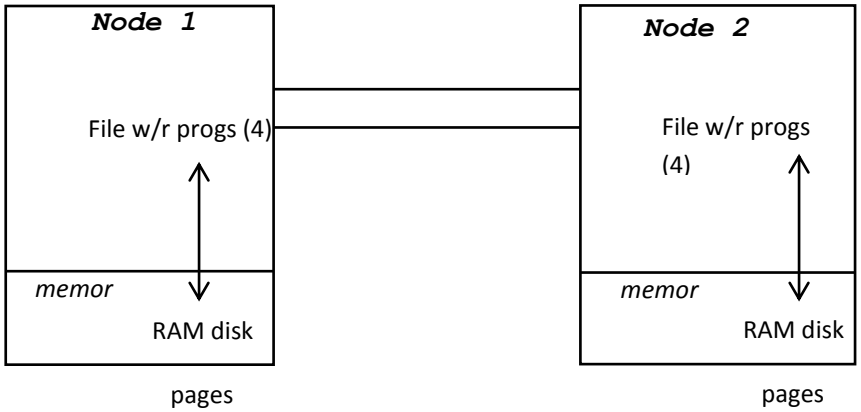
When the RAM disk page accesses were to local NUMA node memory, intense memory activity on the second NUMA node only increased the RAM disk file write and read accesses by 5%. This test scenario demonstrates the benefits of localizing RAM disk accesses on a NUMA system when memory activity is occurring on other NUMA nodes.

Test Scenario 2

In this test scenario:

- Two RAM disks were used: one disk's pages resided in the first NUMA node, and the second disk's pages resided in the second NUMA node.
- An *ext3* file system was created in each RAM disk and both *ext3* file systems were mounted.
- Four copies of the file write/read program were executed on each of the two NUMA nodes, for a total of eight copies simultaneously executing on the system.
- Each of the eight copies of the program wrote and read to their own 2MB file that was located on one of the mounted RAM disk file systems. Each write and read was 2MB in size.

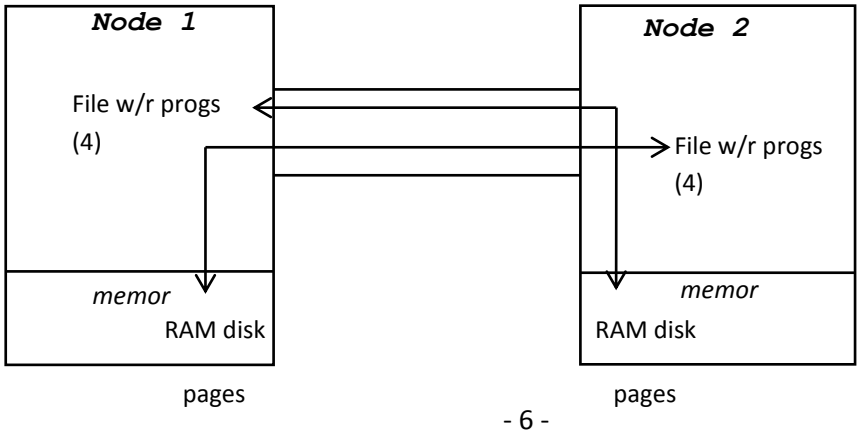
Local RAM Disk Files



In this first case, all of the eight file write/read program's RAM file pages resided in local NUMA node memory. The highest timings observed were:

each lseek and write call:	6.968 milliseconds
each lseek and read call:	7.050 milliseconds

Remote Memory RAM Disk Files



In the second case, all of the eight file write/read program's RAM disk pages resided in the remote NUMA node's memory. The highest timings observed in this case were:

each lseek and write call:	8.301 milliseconds
each lseek and read call:	8.207 milliseconds

Performance Difference

When the RAM disk *ext3* file system files are being remotely accessed, the writes took 19% longer, and the reads took 16% longer.

Summary

The performance measurements discussed in this paper have shown that using RedHawk Linux memory-shielded RAM disk support to place RAM disk pages in the same NUMA node where the RAM disk is most frequently accessed can result in higher performance and reduced memory contention.

About Concurrent Computer Corporation

Concurrent (NASDAQ: CCUR) is a global leader in innovative solutions serving aerospace and defense; automotive; broadcasting; cable and broadband; financial; IPTV; and telecom industries. Concurrent Real-Time is one of the industry's foremost providers of high-performance real-time computer systems, solutions, and software for commercial and government markets, focusing on areas that include hardware-in-the-loop and man-in-the-loop simulation, data acquisition, industrial systems and software. Operating worldwide, Concurrent provides sales and support from offices throughout North America, Europe, Asia and Australia. For more information, please visit Concurrent Real-Time Linux Solutions at http://real-time.ccur.com or call (800) 666-4544.