A Concurrent Real-Time White Paper

2881 Gateway Drive
Pompano Beach, FL 33069
(954) 974-1700
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

アブストラクト

このホワイトペーパーでは、RAM ディスクデータの NUMA ノード上のページ配置を制御する RedHawk Linux の機能を説明します。

RAM ディスクページを RAM ディスクアプリケーションを実行する同じ NUMA ノードに置くことは、より効率が良く、システム全体のメモリコンテンションを削減する効果をもたらすことができます。

このペーパーでは適切な RAM ディスクページ配置の利益を示すいくつかのテストケースのシナリオを記述します。

概要

非ユニフォームメモリアーキテクチャ(NUMA)コンピュータシステムでのメモリアクセス時間は、プロセッサの相対的なメモリ位置に依存します。 NUMA マルチプロセッサシステムは2つ以上の NUMA ノードで構成されています、そしてそれぞれのノードは1つ以上のプロセッサとメモリを持っています。 NUMA システムでは、すべてのメモリをすべてのプロセッサがアクセスできます。 メモリへのアクセスは、ローカル、あるいはリモートで行われます。– プロセッサと同じノードの上に位置するメモリへのローカルなアクセスとプロセッサが存在するところと異なったノードに存在するメモリへのリモートアクセス。 リモートアクセスは、リモートメモリをアクセスしているプロセッサが NUMA ノードバス相互接続上で離れた位置にあるとき、レイテンシィ(潜伏期)と増加したバスコンテンションが特に増加します。

歴史的に、 RedHawk Linux は、NUMA ノードにメモリ – シールドの機能を提供しています。メモリシールドされた NUMA ノードの上でだけ実行を割り当てられたアプリケーションに属しているユーザー空間のページがそのノードのメモリに存在するでしょう。 RedHawk のメモリシールド機能は、メモリシールドされたアプリケーションが行うローカルなメモリアクセスの量を増やして、ノードをまたがるメモリバスコンテンションの量を減らします。

RAM ディスクは、それがディスク・ドライブであるかのように、使われる RAM / メモリの1部です。 RAM ディスクはサイズを固定して、そして通常のディスクパーティションのように振る舞います。 アクセス時間は物理的なディスクと比較して RAM ディスクの方がずっと高速です。 システムの電源をダウンするとき、 RAM ディスクにストアされたデータは失われます;この理由から RAM ディスクは高速のアクセスを必要とする一時的なデータをストアするための良い方法として利用されます。

RedHawk6.0 では、RAM ディスクのページが存在する NUMA ノードを指定する機能が加えられました。 ユーザーは、 RAM ディスク毎に、 RAM ディスクのメモリデータを保つ NUMA ノードをコントロールすることが出来ます。 この新しい機能はさらにメモリ – シールドアプリケーションのパフォーマンスを高めることができます。

RAM ディスク NUMA ノードインタフェース

ユーザーが / proc ファイルインタフェースを使用して RAM ディスクの NUMA ノードページ配置を指定することができます。 さらに、RAM ディスクのページカウント値をノード毎に評価することが出来ます。

/proc/driver/ram/ram[n] /nodemask ファイルは RAM ディスクページが存在する NUMA ノードセットするために使うことができ、現在の状態を見ることが出来ます（この nodemask は16進の bitmask 値です）。

デフォルトでは、 RAM ディスクページはすべての NUMA ノードの上に存在することが出来ます。例えば、

8 NUMA ノードシステム上の RAM ディスク2のデフォルト nodemask は以下のようになっているでしょう:

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

コマンドは nodemask 値をセットして、そしてチェックすることができます:

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

RAM ディスクページが RAM ディスクに割り当てられた後、 /proc/driver/ram/ram[n]/pagestat ファイルは node 毎のページカウント値を表示する事が出来ます。 例えば:

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

パフォーマンス測定

このペーパーはリモート NUMA ノードで存在する RAM ディスクページと比較して RAM ディスクにアクセスしているプロセスにローカルな RAM ディスクページを使うことについての潜在的な公演と決定論利点を示す2つのテストのシナリオを論じます。 RedHawk Linux のメモリシールド RAM ディスクサポートは RAM ディスクページの NUMA ノード配置をコントロールするこれらのテストで使われます。

テストのシナリオは、必ずしも実際のアプリケーションの典型的なメモリと RAM ディスク使用方法ではありませんが、これらのテストのシナリオで観察されたいくつかの相違の部分が実際の実世界のアプリケーションで見られるでしょう。 テストは4プロセッサ Opteron – 4つの NUMA ノードとノード毎に4つのコアを持っているシステム 上で実行されました。 下記のテストはシステムの4つの NUMA ノード上の2つを利用しました。

テスト方法

ファイル書き込み / 読み取りプログラムが2つのテストのシナリオの両方ともで使われました。このプログラムは/dev/ram[n] ファイルを直接、あるいはすでにマウントされた RAM ディスクファイルシステムの中ファイルいずれかを開きます. RAM ディスクアクセスがファイルシステムページキャッシュを迂回して、そしてユーザースペースでユーザーのバッファの間の直接のファイルデータコピーを、そしてカーネルスペースに RAM ディスクページをもたらすように、 O_DIRECT オープン(2)フラグは使われます。

プログラムは、オープンしたファイル上で最初に clock_gettime(2) を使った開始時間を得て、次に lseek(2)と write(2)、あるいは lseek(2)と read(2)のいずれかを実行し、ループします。 システムサービスコールのこのセットを実行した後で、 clock_gettime (2)が再び終了の時間を得るために使われます。

lseek の擬似コードと書き込みループは下記のようになります:

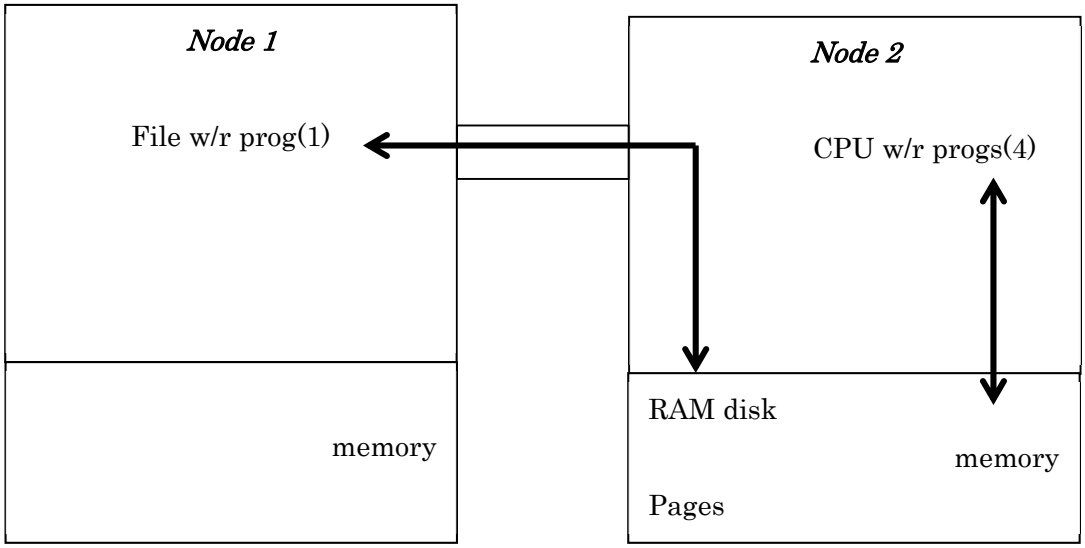
```
clock_gettime(CLOCK_MONOTONIC, start);
for (loop = 0; loop < loopcount; loop++) {
lseek(fd, 0, SEEK_SET);
write(fd, buffer, size);
}
clock_gettime(CLOCK_MONOTONIC, end);
```

この RAM ディスク write/read ループテストに加えて、ユーザースペースメモリバッファから連続的に読み書きする CPU write/read メモリテストがCPUメモリアクセストラフィックを生成するために下記の最初のテストのシナリオで使われました。 このテストはこれらのCPUメモリアクセスが、実際の物理的なメモリ(キャッシュされない)アクセスをもたらして、CPUキャッシュミスを起こす傾向があるように書かれています。

テストシナリオ1

このシナリオでは、1つのファイル書き込み / 読み取りプログラムが8MBが生の RAM ディスク装置に直接書いて、そして読むプログラムが /dev/ram0 を開いて、そしてコマンドを発行した最初の NUMA ノードの上で実行されました。

リモート RAM ディスクページを使用



初めに、 RAM ディスクは2番目の(リモート) NUMA ノードに位置するようにセットアップされました。

リモートノードの上で他のアクティビティが無い場合に、RAM ディスクにアクセスした時間を以下に示します:

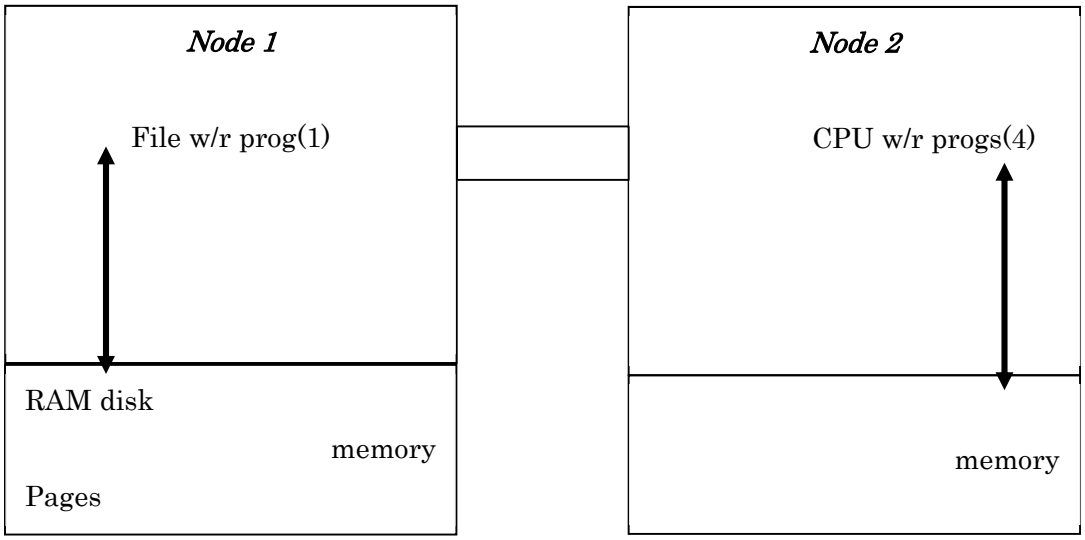
each lseek and write call: 8.916 milliseconds
each lseek and read call: 8.700 milliseconds

そして次に RAM ディスクページを割り当てたリモート NUMA ノード 2 上で、CPU read/write メモリプログラムの4つのコピー(CPU毎に1つのコピー)を実行し、RAM ディスクページにアクセスを増加させた場合の時間を以下に示します:

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

結果、リモート RAM ディスクページのアクセスが、このテストのシナリオの write 呼び出しで 53%、read 呼び出しで 49%、コンテンションが増加しました。

ローカル RAM ディスクページを使用



今度は、RAM ディスクページはローカルな NUMA ノードに配置しました。
シングルファイル書き込み / 読み取りプログラムは、最初と同じ NUMA ノード 1 の上で実行しました。

ノード2の上で他のアクティビティが無い場合に、RAM ディスクにアクセスした時間を以下に示します:

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

そして次に RAM ディスクページを割り当てたリモート NUMA ノード 2 上で、CPU read/write メモリプログラムの4つのコピー(CPU毎に1つのコピー)を実行した場合の時間を以下に示します:

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

RAM ディスクのページアクセスがローカルな NUMA ノードメモリで行われたとき、2番目の NUMA ノード上のメモリアクティビティは RAM ディスクファイルの write と read を 5%増加させただけでした。

このテストシナリオは、メモリアクティビティが他の NUMA ノードの上で発生しているとき、NUMA システム上で RAM ディスクのアクセスを局地的に制限する利益を示します。

テストシナリオ2

このテストのシナリオ:

● 2つの RAM ディスクを使用:1 番目のディスクは、1番目の NUMA ノード 1 に割付けました、そして2番目のディスクは2番目の NUMA ノード 2 に割り付けました。

● ext3 ファイルシステムをそれぞれの RAM ディスクに作成しました、そして両方の ext3 ファイルシステムをマウントしました。

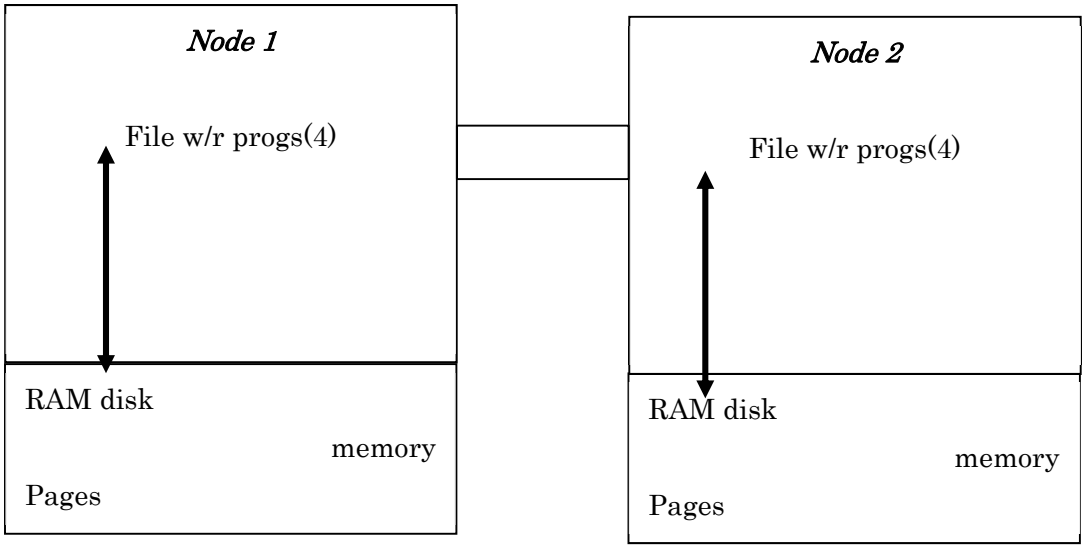
● ファイル write/read プログラムの4つのコピーを、2つの NUMA ノードのそれぞれの上で実行し、システムで合計8つのコピーを同時に実行しました。

●それぞれがマウントされた RAM ディスクファイルシステム大きさは 2MB で、プログラムの8つのコピーのそれぞれが 2MB のファイルを write し read します。

このテストのシナリオで、8つのファイル read/write プログラムが実行された間に、追加のCPUあるいはメモリアクセスアクティビティはシステムに加えられませんでした。

下記に示されたファイルアクセス 時間における相違は厳密にローカル対リモート RAM ディスクファイルシステムアクセスの相互作用のためです。

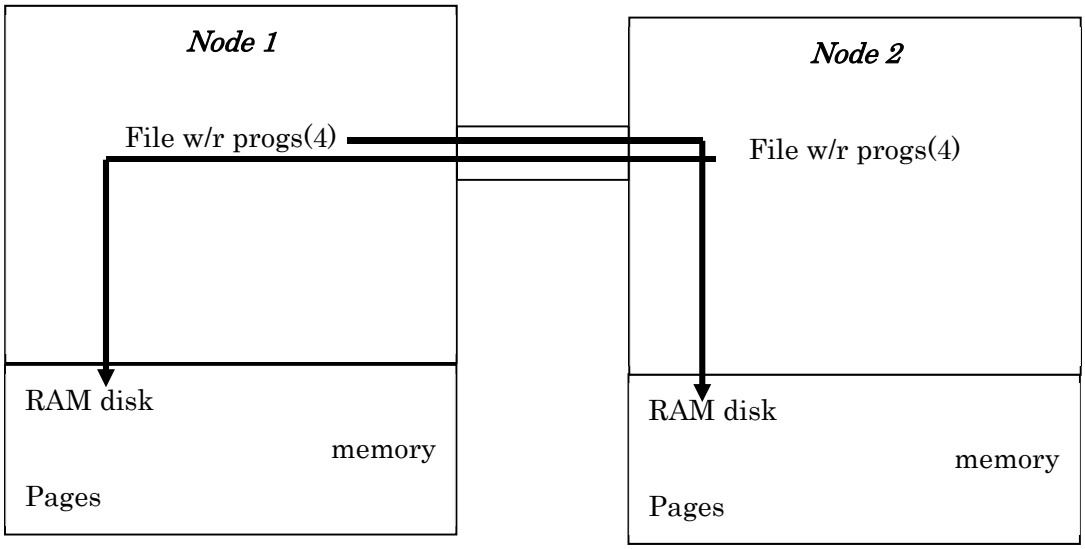
ローカル RAM ディスクファイル



この最初のケースでは、8つのファイル write/read プログラムの RAM ファイルページのすべてがローカルな NUMA ノードメモリに存在します。

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

リモート RAM ディスクファイル



2回目のケースでは、8つのファイル write/read プログラムの RAM ディスクページのすべてがリモート NUMA ノードのメモリに割り付けられました。 この場合に観察された最も高い 時間を以下に示します:

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

パフォーマンスの相違

ext3 ファイルシステム RAM ディスクファイルを間接的にアクセスしているとき、書き込みは 19% 長くかかりました、そして読み取りは 16%長くかかりました。

要約

このペーパーで論じられたパフォーマンス測定で は、RedHawk Linux の RAM ディスクシールドを使って RAM ディスクページを RAM ディスクを最もアクセスする同じ NUMA ノードに置くというアプローチが、より効率が良い、そしてメモリコンテンションを減少させることを示しました。