

kobayashikorio@gmail.com

2017/3/30

Mathematica ver.10.3

OOP Cluster Computing

目的

一台のMathematica計算コントローラーに、複数のMathematica計算サーバーを結合して、インタラクティブなMathematica計算クラスター環境を形成する。

手段

OOPにより全体を構成する。

計算サーバーはインスタンスであり、これにメッセージを配信して計算を実行する。

インスタンスの要件はアソシエーションにより与える。

コントローラーとサーバーとはnc(netcat)を用いたソケットより結合する。

計算サーバーには、ハードウェア要件を簡素化するために、USB-OTGが使える

Raspberrypi Zeroを用いた。

Raspberrypiの電源に係る安定性を考慮して、計算サーバーの立ち上げは自動化していない。

準備

RNDIS(Remote Network Dirver Interface Specification) Ethernet Protocol over USB の使えるRaspberryPi Zeroを複数台用意し、これらを繋ぐUSB Data Hubを準備する。

RaspberryPiに一意的なnode-nameを付ける (raspberrypi1,raspberrypi2など)。

Macのhost-nameを調べる (uname -n)。

RaspberryPiの各計算サーバーにサーバープログラムをnanoエディタなどを用いてインストールする。プログラムファイル名はinitであり、内容は以下の通り(ソケット番号は各計算サーバー毎に異なる。またchmod +x initしておくこと)。

```
$ cat init
While[True,
Run["nc -l 8000>input"];
temp=ReleaseHold[<<input];
temp >>output;
Run["nc kobayashi-no-MacBook-Air-2.local 8002<output"]
```

各Raspberrypi計算サーバーのinitファイル中のポート番号は、互いにバッティングしないように付ける。

outputポートのノード名は、コントローラーのhost-nameであり、この場合”kobayashi-no-MacBook-Air-2.local”である。

sshで接続した、各raspberrypi計算サーバーにおいて、Mathematicaを立ち上げる

```
$ wolfram <init &
```

設定をチェックするためにMacから以下を実行してみる(計算サーバーがraspberrypi.localの場合)。

```
$ nc -l 8002 >output|nc raspberrypi.local 8000 <<EOF
> 10!
> EOF
```

```
$ cat output
```

```
3628800
```

あるいは、名前付きパイプを作っておいて

```
$ mkfifo npipe
```

```
$ nc -l 8002 > npipe | nc raspberrypi.local 8000 << EOF | cat npipe
```

*LinkCreateはMathematica V10 <-> *Mathematica* RaspberryPiの間ではうまく働かない

*以下の方法は動作する

```
$ nc -l 2222
socket=SocketConnect["kobayashi-no-MacBook-Air-2.local:2222"];
str=ReadString[socket]
```

備考

LinkCreateはMathematica V10 <-> *Mathematica* RaspberryPiの間ではうまく働かない。

以下の方法は動作する。

```
$ nc -l 2222
socket = SocketConnect["kobayashi-no-MacBook-Air-2.local:2222"];
str = ReadString[socket]
```

プログラムの実行

ディレクトリをセット

```
SetDirectory[NotebookDirectory[]];
```

ソケットの通信プロセスをスタティックに記述する

```
com1="nc -l 8002 >output1 |nc raspberrypi.local 8000
```

```
<input1";
com2="nc -l 9002 >output2 | nc raspberrypi1.local 9000
<input2";
com3="nc -l 9502 >output3 | nc raspberrypi2.local 9500
<input3";
```

オブジェクトの要件を連想により記述する

```
obj={
  <| "name"->node1, "comm"->com1, "in"->"input1", "out"-
>"output1", "p"->{2000,3500}|>,
  <| "name"->node2, "comm"->com2, "in"->"input2", "out"-
>"output2", "p"->{3501,4000}|>,
  <| "name"->node3, "comm"->com3, "in"->"input3", "out"-
>"output3", "p"->{4000,4500}|>;
```

計算サーバークラスの定義。この例では、メルセンヌ素数を計算する

```
new[nam_] := Module[{ps, pe},
  mersenneQ[n_] := PrimeQ[2^n - 1];
  setv[nam[{s_, e_}]]^:={ps, pe}={s, e};
  calc[nam]^:=Select[Range[ps, pe], mersenneQ]
];
```

インスタンスを生成する（連想を使う時には、純関数に連想キーの名前を結合できる）

```
Map[new[#name]&, obj];
```

objectを計算サーバーに書き込む（インスタンスのイメージがSaveでダンプされる）

```
Map[Save[#in, #name]&, obj];
Map[Run[#comm]&, obj];
```

各計算サーバーにおかれたインスタンスに対して、メッセージを送る（setv関数を実行する）

```
Map[Put[Hold@setv[#name[#p]], #in]&, obj];
Map[Run[#comm]&, obj];
```

関数（calc関数）を各計算サーバーで実行する
calc関数は実行時間がかかるのでプロセス化する

```
Map[Put[Hold@calc[#name], #in]&, obj];
proc=Map[StartProcess[{$SystemShell, "-
c", #comm}]&, obj]
{ProcessObject[0], ProcessObject[1], ProcessObject[2]}
```

プロセスが終了するのを待ち合わせる

```
Map[ProcessStatus[#]&,proc]  
  {Finished,Finished,Finished}
```

結果を読み込む

```
Map[FilePrint[#out]&,obj];  
  {2203, 2281, 3217}  
  {}  
  {4253, 4423}
```

この後も、インスタンスを再度生成することなく、計算範囲を変更して（連想キーp）、計算サーバーによる並列計算を何度も実行できる。

メルセンヌ素数は、...,1279,2203,2281,3217,4253,4423,9689,...である。