

Bot と Wiki を使った試験的な並列プログラミング

山之上 卓^{†1}

概要: Bot と Wiki を使った試験的な並列プログラミング環境およびプログラム例を示す。情報セキュリティ担当者が頭を悩ませていた悪性 Bot の耐障害性と超並列性を、科学技術計算や一般的な計算を行うために有益な方向に利用することを目指す。例として動的計画法を用いて最小経路問題を解く並列プログラムを示す。ここで、必要な計算資源 (Bot と Web ページの数) はノード数に比例し、最小経路を計算するのに必要な時間は、求まる最小経路の弧の数に比例する。

キーワード: Wiki, Bot, parallel programming, dynamic programming

Experimental Parallel Programming Using Bots and Wiki Software

TAKASHI YAMANOUE^{†1}

Abstract: An experimental parallel programming environment, which uses Bots and Wiki software, and an example program of the environment are discussed. The environment aims to use high availability and massively parallel feature of malicious bots for beneficial purposes. Parallel dynamic programming for solving a minimal path problem is shown as an example. Resources such like the number of bots and web pages are proportional to the number of nodes, and the time to solve the problem is also proportional to the number of arcs of the minimal path.

Keywords: Wiki, bot, parallel programming, dynamic programming

1. はじめに

Bot と Wiki を使った試験的な並列プログラミング環境およびプログラム例を示す。情報セキュリティ担当者が頭を悩ませていた悪性 Bot の耐障害性と超並列性を、科学技術計算や一般的な計算を行うために有益な方向に利用することを目指す。例として動的計画法を用いて最小経路問題を解く並列プログラムを示す。ここで、必要な計算資源 (Bot と Web ページの数) はノード数に比例し、最小経路を計算するのに必要な時間は、求まる最小経路の弧の数に比例する。

我々は Arduino と Android と Wiki を組み合わせた M2M システムを開発している [6]。ここで Arduino と Android を組み合わせたセンサ-アクチュエータ側端末は一種の Bot であり、Wiki に記載された設定やプログラムによって動作する。また、この Bot の一部を改変することにより、情報セキュリティを脅かす悪性 Bot の所在を検知したり悪性 Bot の通信を横取りしたりして、悪性 Bot を捕まえるシステムの開発も行っている [7]。これらの Bot のプログラミング環境は、ノイマン型コンピュータがプログラムもデータもメモリ上に記述されているのと同様に、プログラムもデータも Wiki ページ上に記述される。これを「Wiki ページ型プログラミング」と呼ぶこととする。我々は Wiki ページ型プログラミングの特徴を使い、可用性の高い分散シス

テムの構築の可能についても検討を行っている [9]。

今回、我々の Wiki ページ型プログラミングによって動作する Bot を、Single Program, Multiple Data (SPMD) [10] 型の並列プログラミングが行えるように拡張した。この並列プログラミングのことを「Wiki ページ型並列プログラミング」と呼ぶこととする。Wiki ページ型並列プログラミングの場合、それぞれのボットは従来型の逐次処理を行うが、全体としてはデータフロー型並列処理を行う。

2. 可用性向上と SPMD

Wiki ページ型並列プログラミングにおいては、Bot net の C&C サーバに相当する Wiki サーバが停止しても、Bot そのものは動作を継続できる。また、もし Bot が複数いた場合、Bot の 1 台が停止しても別の Bot が動作を継続できる。この性質を使って、システムの可用性能力を高める方法を検討する (図 1)。

2.1 前提

Bot は定期的に Wiki サーバのページに書かれたスクリプトとデータを読み込み、一定時間内にスクリプトの実行を終了し、定期的に実行結果や Bot の活動状況を Wiki サーバに書き込むものとする。また、Bot はそれぞれを識別できる ID を持つものとする。

2.2 オブジェクト指向言語の継承の利用

Wiki ページの読み書きを行う「An Inter-Wiki Page Data

^{†1} 福山大学
Fukuyama University

Processor for a M2M System」[6]では、おなじような動作を行うスクリプトが書かれた Wiki ページすべてに、ほぼ同じスクリプトを記述していた。オブジェクト指向プログラミング[3]の「Class」と「継承」を利用することにより、このような無駄な記述の削減を行うことが可能である。Bot が直接読み込む Wiki ページを「Object ページ」と呼ぶことにする。Object ページが利用する、スクリプトを「Class」と呼ぶことにする。複数の、似た動作をする Object ページが、似た部分の動作を記述した共通の Class を利用することでスクリプトの記述量が削減される。この Class により SPMD (Single Program, Multiple Data)型の並列プログラミングが可能になる。

Object ページを Bot が読み込むとき、そこに書かれた include 命令に従って Class のページ(Class ページ)を読み込み、「継承」を行う。Class ページと Object ページの間で、重なった代入やプログラムが存在した場合は、Object ページの記述を上書きする。これによって、オブジェクト指向言語におけるクラスの継承と同様の効果が得られる。

Class ページや Object ページで利用できるコマンドやプログラミング言語については、文献に記載されたものを拡張したものが利用できる。

2.3 Class ページのサーバの failover

Class ページを格納した Web サーバで障害が発生した場合に備えるため、同じ Class ページを格納した Primary サーバと Secondary サーバを準備しておく。Object ページが Class を継承しようとしたとき、Primary サーバにアクセスできないときは、Secondary サーバにアクセスして Class を継承する。これを実現するため、Object ページでは、

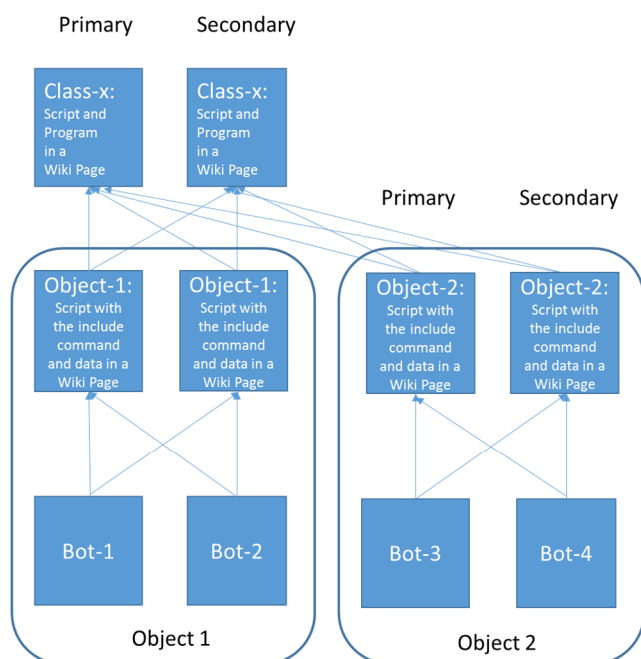


図 1. 可用性向上

Figure 1. Enhance resilience.

“include <class-page-1> or <class-page-2>”

のようなコマンドが利用できるようにする。<class-page-1>と<class-page-2>は Class が記述された Wiki ページの URL であり、最初の URL の読み込みに失敗したら次の URL を読み込む。

2.4 Object ページのサーバの failover

Object ページを格納した Web サーバで障害が発生した場合に備えるため、Object ページについてもそれを格納する Primary サーバと Secondary サーバを用意する。Bot に対して、Primary サーバにアクセスできない場合に Secondary サーバにアクセスするよう指示するため、Object ページでは、

“object page <object-page-1-1> or <object-page-1-2>”

のようなコマンドが利用できるようにする。<object-page-1-1>を実行していた Bot が<object-page-1-1>に実行を書き込むとして失敗したら、<object-page-1-2>に実行結果を書き込む。「An Inter-Wiki Page Data Processor for a M2M System」では、読み込んだ Wiki ページのスクリプトもデータもすべて一旦 Bot が保持していて、実行結果を書き込むときは、これらの保持されていた部分に実行結果の部分を上書きしたページを作成し、そのページごと、書き込みを行っていた。この性質を使うことで、<object-page-1-1>を実行していたとき、<object-page-1-1>にアクセス不能になったら、Bot は<object-page-1-1>にもともと書き込まれていた実行前データも含めて、<object-page-1-2>に書き込みを行う。これによって、入力データも含めて failover が可能になる。<object-page-1-1>が復活したら、同様にして <object-page-1-1>にデータを移行し、実行が継続される。

2.5 Bot の failover

Bot に障害が発生した場合に備えて、1つの Object ページに対して複数の Bot を待機させ、もともと動作していた Bot が停止した場合、待機していた Bot を実行させるようにする。これを実現するため、Object ページでは、

“device <Bot-1> or <Bot-2> start after no write for <time>”

のようなコマンドが利用できるようにする。ここで動作している Bot は定期的に、それが動いていることを示すため、その Bot の ID とその時間を Object ページのデータ部分に書き込むこととする。Bot は自分の ID が <Bot-1>か<Bot-2>であることを確認してページの読み込みを行い、(副作用がない場合は)実行する。もし、Object ページのデータ部分に書き込まれた Bot の ID が自分で、書き込まれた時間と現在の時間の差が<time>以下であれば、自分が書き込みを行う。もし、自分の ID が<Bot-1>か<Bot-2>のどちらかで、Object ページのデータ部分に書き込まれた ID が自分でなく、かつ、そこに書かれた時間と現在の時間の差が<time>を超えたら、自分が書き込みを行う。

2.6 システム更新への応用

この Bot を使った応用システムそのもののシステム更新を、その運用停止時間をできるだけ短くして実施できると

嬉しい。これは Primary の Class ページが実行されているときに、Secondary の Class ページを新しいものに入れ替え、その後、強制的に Primary の Class ページを保持しているサーバを停止し、Class ページのサーバの failover が実行されたのちに、Primary の Class ページを新しいページに更新することで実施できる。Object の記述の更新についても同様のことを行えば良い。どちらの場合も事前に動作確認テストが必要である。

2.7 Bot の一斉操作

多くの Bot を一斉に操作する場合、それぞれの Bot に対応した Object ページを個別に書き換えて制御するのは大きな労力が必要となる。Class ページを使うことにより、多くの Bot を一か所から一斉に操作することが可能になり、このような労力を削減できる。

複数の Bot に対応した複数の Object ページが 1 つの Class ページを include している場合、それぞれの Bot は定期的に同じ Class ページを読み、実行する。Class ページに書かれたコマンドやプログラムはそれを使うすべての Bot で実行される。この Class ページが Object ページの更新を行う記述を持たないか、または、Class ページの Object ページを更新する記述が、対応する Bot によって実行されなければ、Object ページが書き換えられることはなく、実質的に、すべての Bot の動作が止まったことになる。Class ページの、Object ページを書き換えるような記述が対応する Bot によって実行されれば、すべての Bot の動作が行われたことになる。従って、ひとつの Wiki ページ(バックアップも入れれば 2 つ)の記述を変更することで、複数の Bot の振る舞いを一斉に制御することができる。

2.8 Object ページのデータ

Wiki ページ型プログラミングはデータも Wiki ページに記述される。データは Object ページのコマンドやプログラムの列の後に、“result:”の行を書いて、その行の後に記述する。新たに書き込まれるデータもこの部分に記述される。

2.9 Class ページと Object ページの簡単な例

簡単な例として、図 2 に簡単な Class ページの例、図 3 にこのクラスを使う Object ページの例の実行前、図 4 にその Object ページの実行後を示す。

図 2 のクラスは、1+2 を計算して、result: の下にその結果を書くプログラムである。プログラムは、その上下を

```
command: program <プログラム名>
```

と

```
command: end <プログラム名>
```

で囲んで記述する。

```
command: run <プログラム名>
```

は、この行で<プログラム名>のプログラムが実行されることを表す。

プログラムは左に

```
program:
```

を書いて表す。

このプログラム中の

```
ex("service","clear sendBuffer")
```

は、Object ページの result: 以下の行に書き込まれる内容を格納する “sendBuffer”を空にすることを表す。

```
ex("service","putSendBuffer "+x)
```

は、sendBuffer の内容の最後に、文字列 x を加えることを表す。

```
ex("service","sendResults.")
```

は sendBuffer の内容を Wiki ページの result: の下書き込むことを表す。



図 2. 簡単な Class ページ の例

Figure 2. An example of a simple Class page



図 3. 実行前の Object ページの例

Figure 3. An example of an Object page, before the execution.



図 4. 実行後の Object ページの例

Figure 4. The Object page of the Figure 3, after the execution.

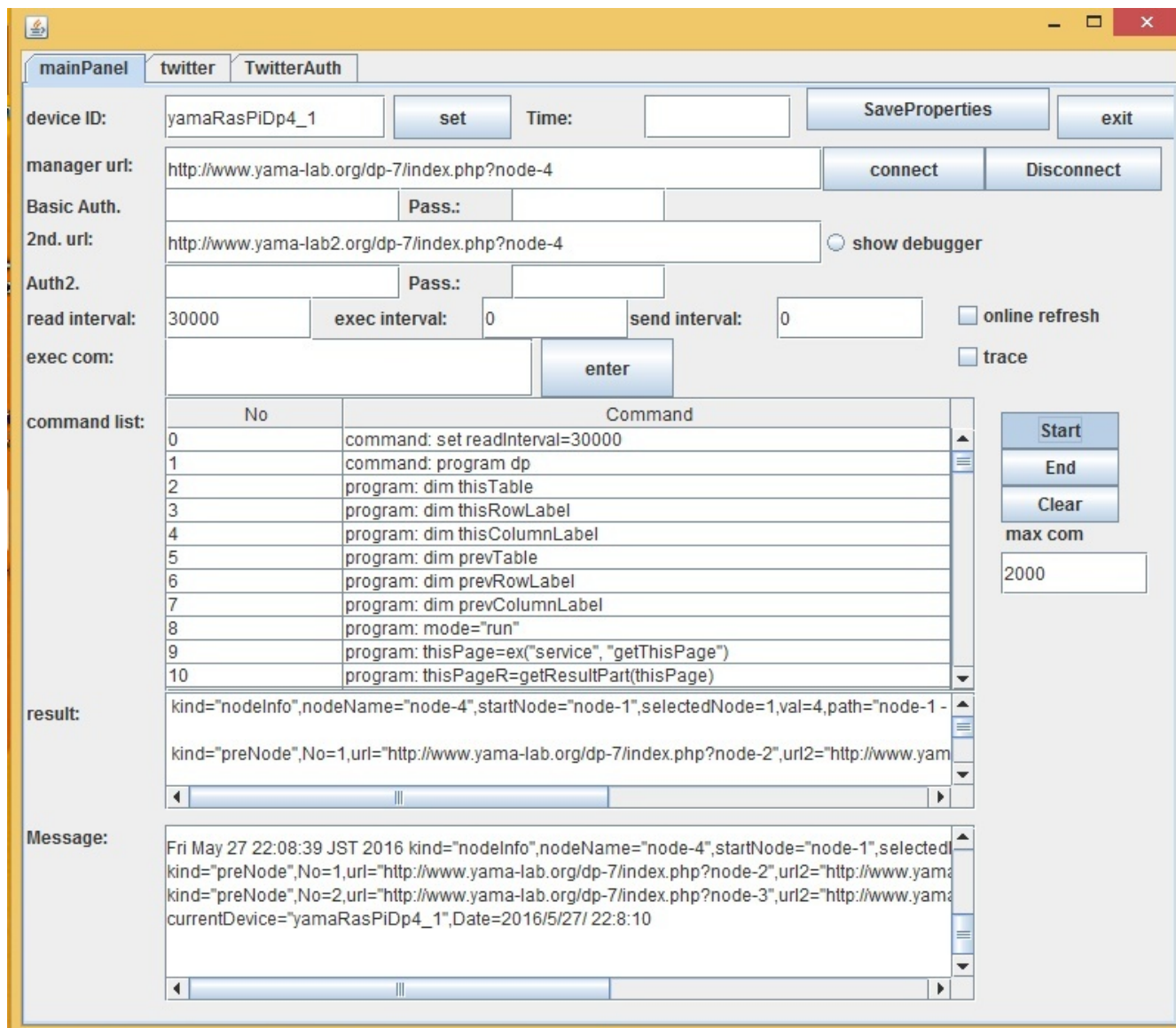


図 5. ボットの GUI

Figure 5. GUI of the Bot.

図 4 の 3 が 1+2 の実行結果である。最後の行の
currentDevice="dev1",Date=2016/5/29/ 12:16:38
は、Bot の側で障害が発生した場合に 他 の Bot で failover
を行うのに必要な、このページを最後に実行した Bot の
ID と、その日時を表している。

3. Bot の詳細

図 5 に Bot の GUI を示す。

Manager url の欄に、コマンドが書かれた Wiki のペー
ジの URL を入力する。Start ボタンをクリックすると、Bot
は、read Interval に書かれた間隔でその URL にある Wiki
のページを入力し、そこに書かれたコマンドやプログラム
を実行する。Manager url の読み込みが失敗した場合は、2nd
url の右の欄に書かれた URL にある Wiki のページを入
力し、そこに書かれたコマンドやプログラムを実行する。

Basic Auth の右の欄には、Manager url のページを読み込む
とき、Basic 認証が必要となったときの ID とパスワードを
記述する。Auth2 の右の欄には、2nd url のページを読み込む
ときに Basic 認証が必要になったときの ID とパスワード
を記述する。2nd url は、Manager url で指定された Object ペ
ージが読み込まれたとき、そこに書かれた

“object page <object-page-1-1> or <object-page-1-2>”
の <object-page-1-2> が自動的に埋められる。

読み込まれたページに書かれたコマンドは、Command
list の欄に上から順番に表示される。Command list の欄に表
示されたコマンドは、exec Interval で示された間隔で、が上
から順番に実行を繰り返す。Exec Interval の値が 0 の場合
は、Object ページが読み込まれてからすぐに、1 度、そのペ
ージに書かれたコマンドとプログラムが実行される。

Bot は、それに付けられたセンサの値を一定時間ごとに

入力し、その値をセンサ名と時刻と共に、バッファに格納し、Send interval の右の欄で示された間隔で、そのバッファに書かれた内容を Object ページの Result: の下書き込む機能を持っている。Send interval の値が 0 であるときは、コマンドやプログラムで陽に指定されない限り、バッファの内容を Object ページに書き込まない。

Message の欄には、コマンド実行時のメッセージが表示される。

この Bot は Java で記述されている。Wiki ページの読み込み部分は、「Pukiwiki で Java プログラムの起動とそのデータ保存を可能とするシステムの試作」[4]や「Wiki と携帯型遠隔操作端末を使った情報セキュリティ対策システム」[5]などで使ったクラスと Factory Method パターンを利用した。コマンドで Twitter の API も利用することが可能で、あり、この実現には Twitter4J[11] を利用した。

4. 動的計画法のプログラム

4.1 4.1 Object ページによる node と arc 情報の記述

ラベル付き有向グラフにおけるスタートからゴールまでの最小経路とその距離を求める Wiki ページ型並列プログラミングの例を示す。ここでは図 6 のような、4 つの node と 4 つの数値ラベル付き arc を持ったグラフの最小経路を求めるプログラムを示す。

(Backup も含めて)2 枚の Object ページを 1 つの node に割り当て、2 つの Bot をそれぞれの Object ページに割り当て、必要な情報を result: の下に記載する。図 7 に Object ページのノード情報の書き方を示す。result: の下の行の、

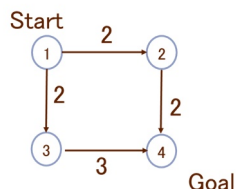


図 6. 簡単な例

Figure 6. A simple example for solving the minimal path problem.

kind="nodeInfo" ...

の行は、この行に node の情報が書き込まれることを表す。<nodeName> は node の名前を表す。<start-node> は Start node を表す。selectedNode =<number>の <number>は、その node に直接至る経路を持つ node が複数あった場合、start-node からその node までの経路(の計算の途中経過)が最も小さくなる node の、このページ上の番号を表す。<val>は最小経路の経路長、<path> は start node からこのノードまで、最小経路を通ったときの、node 名の列を順番に並べたものが記述される。プログラム実行前は、<number>は 0,

<val>は 0, <path> は "" としておく。

kind="preNode", No=n...

の行は、その node に直接至る経路がある node(前 node)が n 個あった場合、n 番目の前 node の情報などを記載している。No= の右の数値は、前ノードの、このページ上の番号である。<previous-node-n-1>, <previous-node-n-2> は、前ノードを表す情報を格納した Object ページの URL を表す。<arcVal-n>は、n 番目の 前 node と、この node の間の弧の長さを表す。<val-n> は start node から前 node までの最小経路長を表す。<path-n>は start node から前 node までの経路を表す。プログラム実行前は、<val-n>は 0, path は""としておく。Start node に対応した Object ページには、kind="preNode" の行はない。

各 Object ページにおいて、node 名は、node-<番号>で図 6 の node を表すことにする。図 7 にプログラム実行前の node-4 に対応した object ページを示す。node-4 は node-2 と node-3 の 2 つの前 node を持っていてこれはそれぞれ、

kind="preNode", No=1...

の行と

kind="preNode", No=2...

の行によって表されている。Node 2 から node 4 に至る弧の値(長さ)は 2 であるため、kind="preNode", No=1 の行の arcVal は 2 になっている。同様に node 3 から node 4 に至る弧の値(長さ)は 3 であるため、kind="preNode", No=2 の行の arcVal は 3 になっている。

```
objectPage <url1> or <url2>
device <device1> or <device2> start after no write for 10 min.
include <class1> or <class2>
result:
kind="nodeInfo",nodeName=<node name>, startNode=<start-node>, selectedNode=<number>, val=<val>, path=<path>
kind="preNode",No=1,url=<previous-node-1-1>,url2=<previous-node-1-2>,arcVal=<arcVal-1>,val=<val-1>,path=<path-1>
...
kind="preNode",No=n,url=<previous-node-n-1>,url2=<previous-node-n-2>,arcVal=<arcVal-n>, val=<val-n>,path=<path-n>
currentDevice=<device>, Date=<date>
```

図 7. Node 情報を記載した Object ページ

Figure 7. Object page for a node



node-4

<http://www.yama-lab.org/dp-7/index.php?node-4>

[トップ] [編集 | 凍結 | 差分 | バックアップ | 添付 | リロード] [新規 | 一覧 | 単語検索 | 最終更新 | ヘルプ]

index

node-3

FrontPage

simple ex
ample

music

NetDraw

```
objectPage http://www.yama-lab.org/dp-7/index.php?node-4 or http://www.yama-lab2.org/dp-7/index.php?node-4
device yamaRasPiDp4_1 or yamaRasPiDp4_2 start after no write for 10 min.
include http://www.yama-lab.org/dp-7/index.php?DpClass1 or http://www.yama-lab2.org/dp-7/index.php?DpClass1
result:
kind="nodeInfo",nodeName="node-4",startNode="node-1",selectedNode=0,val=0,path=""
kind="preNode",No=1,url="http://www.yama-lab.org/dp-7/index.php?node-2",url2="http://www.yama-lab2.org/dp-7/index.php?node-2",arcVal=2,val=0,path=""
kind="preNode",No=2,url="http://www.yama-lab.org/dp-7/index.php?node-3",url2="http://www.yama-lab2.org/dp-7/index.php?node-3",arcVal=3,val=0,path=""
currentDevice="yamaRasPiDp4_1",Date=2016/5/27/ 22:10:40
```

図 8. node-4 を表す Object ページの実行前
Figure 8. Object page for the node-4, before the execution.

```
command: set readInterval=30000
command: program dp
program: dim thisTable
program: dim thisRowLabel
program: dim thisColumnLabel
program: dim prevTable
program: dim prevRowLabel
program: dim prevColumnLabel
program: mode="reset"
program: thisPage=ex("service", "getThisPage")
program: thisPageR=getResultPart(thisPage)
program: ex("service","println "+thisPageR)
program: parseCsv(thisPageR, thisTable, thisRowLabel, thisColumnLabel)
program: lineNumber=getMaxIndex(thisRowLabel)
program: minVal=10000
program: selectedNode=0
program: selectedPath=""
program: preNodes=0
program: for i=0 to lineNumber-1
  各前ノードについての計算
program: next i
program: thisNodeIndex=getIndex(thisTable,thisRowLabel, thisColumnLabel("kind"),"=", "nodeInfo")
program: thisTable(thisNodeIndex,thisColumnLabel("selectedNode"))=selectedNode
program: if mode="reset" then
program: {
  objectPage のノードと Arc の情報の初期化
program: }
program: mode="run" then
program: {
  計算結果によるノードと Arc の情報の更新
program: }
command: end dp
command: run dp
```

図 9. 動的計画法によって最小経路を求める Wiki 型並列プログラミングの class ページ (DpClass1)
Figure 9. Class page (DpClass1) which contains the Wiki page type parallel programming program of minimal path finder.

```

program:  if thisTable(i,thisColumnLabel("kind")) = "preNode" then
program:  {
program:      preNodes=preNodes+1
program:      prevPageUrl=thisTable(i,thisColumnLabel("url"))
program:      prevPage=ex("connector", "getpage "+prevPageUrl)
program:      if prevPage = "ERROR" then
program:      {
program:          prevPageUrl=thisTable(i,thisColumnLabel("url2"))
program:          prevPage=ex("connector", "getpage "+prevPageUrl)
program:      }
program:      if prevPage <> "ERROR" then
program:      {
program:          prevPageR=getResultPart(prevPage)
program:          parseCsv(prevPageR, prevTable, prevRowLabel, prevColumnLabel)
program:          ix=getindex(prevTable, prevRowLabel, prevColumnLabel("kind"),"=", "nodeInfo")
program:          prevVal=prevTable(ix,prevColumnLabel("val"))
program:          arcVal=thisTable(i,thisColumnLabel("arcVal"))
program:          xval=prevVal+arcVal
program:          thisTable(i,thisColumnLabel("val"))=xval
program:          prevPath=prevTable(ix,prevColumnLabel("path"))
program:          thisTable(i,thisColumnLabel("path"))=prevPath
program:          if xval<minVal then
program:          {
program:              minVal=xval
program:              selectedNode=thisTable(i,thisColumnLabel("No"))
program:              selectedPath=prevPath
program:          }
program:      }
program:  }
program:  }

```

図 10.各前 node についての計算

Figure 10. Calculation for each previous nodes.

```

program:  ex("service","clear sendBuffer")
program:  dataline="kind=¥"nodeInfo¥""
program:  dataline=dataline+",nodeName=¥""+thisTable(thisNodeIndex,thisColumnLabel("nodeName"))+¥""
program:  dataline=dataline+",startNode=¥""+thisTable(thisNodeIndex,thisColumnLabel("startNode"))+¥""
program:  dataline=dataline+",selectedNode=0"
program:  dataline=dataline+",val=0"
program:  dataline=dataline+",path=¥"¥""
program:  ex("service","putSendBuffer "+dataline)
program:  for i=0 to lineNumber-1
program:      if thisTable(i,thisColumnLabel("kind")) = "preNode" then
program:      {
program:          dataline="kind=¥"preNode¥""
program:          dataline=dataline+",No="+thisTable(i,thisColumnLabel("No"))
program:          dataline=dataline+",url=¥""+thisTable(i,thisColumnLabel("url"))+¥""
program:          dataline=dataline+",url2=¥""+thisTable(i,thisColumnLabel("url2"))+¥""
program:          dataline=dataline+",arcVal="+thisTable(i,thisColumnLabel("arcVal"))
program:          dataline=dataline+",val=0"
program:          dataline=dataline+",path=¥"¥""
program:          ex("service","putSendBuffer "+dataline)
program:      }
program:  next i
program:  ex("service","sendResults.");

```

図 11. objectPage の node と Arc の情報の初期化

Figure 11. Initializing node and arc information of the Object page

```

program: ex("service","clear sendBuffer")
program: if preNodes=0 then
program: {
program:   thisTable(thisNodeIndex,thisColumnLabel("path"))=thisTable(thisNodeIndex, thisColumnLabel("nodeName"))
program:   thisTable(thisNodeIndex,thisColumnLabel("val"))=0
program: }
program: else
program: {
program:   thisTable(thisNodeIndex,thisColumnLabel("path"))=
program:     selectedPath+ " - "+thisTable(thisNodeIndex, thisColumnLabel("nodeName"))
program:   thisTable(thisNodeIndex,thisColumnLabel("val"))=minVal
program: }
program: dataline="kind=¥"nodeInfo¥"
program: dataline=dataline+",nodeName=¥"+thisTable(thisNodeIndex,thisColumnLabel("nodeName"))+"¥"
program: dataline=dataline+",startNode=¥"+thisTable(thisNodeIndex,thisColumnLabel("startNode"))+"¥"
program: dataline=dataline+",selectedNode="+thisTable(thisNodeIndex,thisColumnLabel("selectedNode"))
program: dataline=dataline+",val="+thisTable(thisNodeIndex,thisColumnLabel("val"))
program: dataline=dataline+",path=¥"+thisTable(thisNodeIndex,thisColumnLabel("path"))+"¥"
program: ex("service","putSendBuffer "+dataline)
program: for i=0 to lineNumber-1
program:   if thisTable(i,thisColumnLabel("kind")) = "preNode" then
program:     {
program:       dataline="kind=¥"preNode¥"
program:       dataline=dataline+",No="+thisTable(i,thisColumnLabel("No"))
program:       dataline=dataline+",url=¥"+thisTable(i,thisColumnLabel("url"))+"¥"
program:       dataline=dataline+",url2=¥"+thisTable(i,thisColumnLabel("url2"))+"¥"
program:       dataline=dataline+",arcVal="+thisTable(i,thisColumnLabel("arcVal"))
program:       dataline=dataline+",val="+thisTable(i,thisColumnLabel("val"))
program:       dataline=dataline+",path=¥"+thisTable(i,thisColumnLabel("path"))+"¥"
program:       ex("service","putSendBuffer "+dataline)
program:     }
program:   next i
program: ex("service","sendResults.");

```

図 13. 計算結果によるノードと Arc の情報の更新

Figure 13. Update the node and arc information by the calculation

計算前の node-4

objectPage <http://www.yama-lab.org/dp-7/index.php?node-4> or <http://www.yama-lab2.org/dp-7/index.php?node-4>
device yamaRasPiDp4_1 or yamaRasPiDp4_2 start after no write for 10 min.
include <http://www.yama-lab.org/dp-7/index.php?DpClass1> or <http://www.yama-lab2.org/dp-7/index.php?DpClass1>
result:
kind="nodeInfo",nodeName="node-4",startNode="node-1",selectedNode=0,val=0,path=""
kind="preNode",No=1,url="http://www.yama-lab.org/dp-7/index.php?node-2",
url2="http://www.yama-lab2.org/dp-7/index.php?node-2",arcVal=2,val=0,path=""
kind="preNode",No=2,url="http://www.yama-lab.org/dp-7/index.php?node-3",
url2="http://www.yama-lab2.org/dp-7/index.php?node-3",arcVal=3,val=0,path=""
currentDevice="yamaRasPiDp4_1",Date=2016/5/28/ 1:0:56

計算終了後の node-4

objectPage <http://www.yama-lab.org/dp-7/index.php?node-4> or <http://www.yama-lab2.org/dp-7/index.php?node-4>
device yamaRasPiDp4_1 or yamaRasPiDp4_2 start after no write for 10 min.
include <http://www.yama-lab.org/dp-7/index.php?DpClass1> or <http://www.yama-lab2.org/dp-7/index.php?DpClass1>
result:
kind="nodeInfo",nodeName="node-4",startNode="node-1",selectedNode=1,val=4,path="node-1 - node-2 - node-4"
kind="preNode",No=1,url="http://www.yama-lab.org/dp-7/index.php?node-2",
url2="http://www.yama-lab2.org/dp-7/index.php?node-2",arcVal=2,val=4,path="node-1 - node-2"
kind="preNode",No=2,url="http://www.yama-lab.org/dp-7/index.php?node-3",
url2="http://www.yama-lab2.org/dp-7/index.php?node-3",arcVal=3,val=5,path="node-1 - node-3"
currentDevice="yamaRasPiDp4_1",Date=2016/5/28/ 2:13:2

図 12. 実行前と実行後の node-4 の Object ページ

Figure 12. Before and after of the calculation of the object page for the node-4

4.2 Class ページによる SPMD の記述

図 9 から図 13 にかけて、に動的計画法の計算を行う SPMD の並列プログラムおよびコマンドを示す。

図 9 の

```
command: set readInterval=30000
```

は、Bot が object ページを読む間隔を 30 秒に設定することを表す。

```
command: program dp
```

はこの行以降にプログラム dp を記述することを表す。

```
program: dim thisTable
```

```
program: dim thisRowLabel
```

```
program: dim thisColumnLabel
```

```
program: dim prevTable
```

```
program: dim prevRowLabel
```

```
program: dim prevColumnLabel
```

は、配列(ハッシュ表)を宣言している。thisTable には、各 object ページの、result: 行以降に記述された、CSV で表された表が格納される。この表は

```
label-1=val-1-1, label-2=val-1-2, ... label-m=val-1-m
```

```
...
```

```
label-1=val-n-1, label-2=val-n-2,...label-m=val-n-m
```

のように、label=値をカンマで区切って並べた行を、n 行並べた形式であることを前提としている。thisRowLabel には、表の行数などが格納される、thisColumnLabel は、表の欄の数と、欄の名前などが格納される。prevTable は、それぞれの object ページが表す node の前 node の情報を格納する。prevRowLabel, prevColumnLabel は、thisRowLabel, thisColumnLabel と同様に、行や列の情報が格納される。

```
program: mode="reset"
```

は、変数 mode に文字列“reset”を代入することを表す。変数 mode の値が“reset”の場合は計算終了後、図 14. の「objectPage の node と Arc の情報の初期化」が実行され、object ページの値を計算前の状態に初期化する。“run”の場合は計算終了後、図 15 の「計算結果によるノードと Arc の情報の更新」が実行される。それ以外の値、例えば“stop”の場合は、bot は Wiki ページに書き込みを行わず、実質的な実行を停止する。

Class ページのこの代入文で代入される文字列を変えることにより、計算を行うすべての Bot を一斉に停止したり、ページを初期化したり、実行を開始したりなど、一か所からすべての Bot を制御することが可能となる。

```
program: thisPage=ex("service", "getThisPage")
```

は、このプログラムが実行される object ページ自身の Wiki の内容を文字列として thisPage に格納する。

```
program: thisPageR=getResultPart(thisPage)
```

は、thisPage に格納された Wiki の内容の中で result: 行以降の内容を thisPageR に格納する。

```
program: ex("service", "println "+thisPageR)
```

は、thisPageR の内容を Bot のメッセージの欄に表示することを表す。デバッグに使用する。

```
program: parseCsv(thisPageR, thisTable, thisRowLabel, thisColumnLabel)
```

は thisPageR に格納された CSV を解析し、thisTable に格納する。

```
program: lineNumber=getMaxIndex(thisRowLabel)
```

は読み込んだ CSV の表の行数を 変数 lineNumber に格納する。

```
program: minVal=10000
```

は、最小の arc の値を初期値として 10000 としている。

```
program: selectedNode=0
```

は、まだ 経路値を最小にする前 node が未定であることを表す。

```
program: selectedPath=""
```

は経路を最小にする前 node が選ばれていないことを示す。

```
program: preNodes=0
```

は、前 node の数を表す変数 preNodes を 0 に初期化している。

図 10 は図 9 の「各ノードについての計算」の部分を表す。この部分は、result: の後に書かれた CSV の中で、kind の欄が、前 node を表す“preNode”である場合に実行される。この部分では、start node から前 node までの最小経路の値と、前 node から自 node までの弧の値(長さ)を加えたものを計算し、その値が最も小さくなる前 node を選んでいる。

```
program: preNodes=preNodes+1
```

は、この部分が実行されるたびに preNodes に 1 を加えることにより、前 node の数を数えている。

以下の

```
program: prevPageUrl=
```

```
thisTable(i,thisColumnLabel("url"))
```

```
program: prevPage=
```

```
ex("connector", "getpage "+prevPageUrl)
```

```
program: if prevPage = "ERROR" then
```

```
program: {
```

```
program: prevPageUrl=
```

```
thisTable(i,thisColumnLabel("url2"))
```

```
program: prevPage=
```

```
ex("connector", "getpage "+prevPageUrl)
```

```
program: }
```

```
program: if prevPage <> "ERROR" then
```

```
program: {
```

の部分は、前 node の object ページを読み込んでいる。Failover を行う為、url の読み込みがエラーになった場合、url2 を読み込んでいる。

```

program: prevPageR=getResultPart(prevPage)
program: parseCsv(prevPageR, prevTable,
    prevRowLabel, prevColumnLabel)
program: ix=getindex(prevTable,prevRowLabel,
    prevColumnLabel("kind"), "=", "nodeInfo")

```

の部分では、前 node の情報を prevTable に読み込み、そのページの中で、nodeInfo の行を変数 ix に入れている。

```

program: prevVal=prevTable(ix,prevColumnLabel("val"))
program: arcVal=thisTable(i,thisColumnLabel("arcVal"))
program: xval=prevVal+arcVal

```

の部分では、変数 prevVal に、その前 node の val の値、すなわち、start node から前 node までの最小経路の値を代入し、変数 arcVal にその時の経路、変数 xval に、その前 node を経由して自 node に至った場合の経路の値を代入している。

```

program: thisTable(i,thisColumnLabel("val"))=xval
program: prevPath=prevTable(ix,prevColumnLabel("path"))
program: thisTable(i,thisColumnLabel("path"))=prevPath

```

の部分では、xval の値を、thisTable の i 行(自 node の表の、今見ている前 node の行)の val の欄に代入し、前 node の path の値を、thisTable の i 行の path の欄に代入している。

```

program: if xval<minVal then
program: {
program:     minVal=xval
program:     selectedNode=
        thisTable(i,thisColumnLabel("No"))
program:     selectedPath=prevPath
program: }

```

の部分では、前 node の中で、xval の値が最小となる前 node の番号を選んで、selectedNode に代入し、そのときの path を selectedPath に代入している。

図 12 は、Goal node である node-4 に対応した Object ページの、プログラム実行前と、実行後 (Minimal path が求まった後) の内容を示す。

5. 計算時間の測定実験

4 章で示した Wiki ページ型並列プログラミングによる動的計画法では、start node から goal node まで、最小値の path の選択が繋がったとき、最小経路とその値が求まる。このときの時間は、もし、各ノードにおける計算時間が同じであれば、最小経路の弧の数に比例する時間となるはずである。従って、Object ページの読み込み間隔を TI、その object ページにおいて、ページが読み込まれてから計算が終了するまでの時間を TX とし、最小経路の時の弧の数を Na とすると、平均的な計算時間 T は、

$$T=Na(TI/2+TX)$$

に近い値になることが予測できる。TI が 30 秒であり TX が TI に比べて無視できるほど小さい場合は、T は最小経路の

arc 数に 15 を掛けた値になることが予測される。

これを確かめるため、最小経路が 2, 4, 8 となるような場合について、最小経路を求める計算時間を測定した。この計算に使ったグラフを図 16 に示す。また測定の様子を図 17 に示す。なお、ここでは failover 起らないことを仮定し、backup のための Wiki ページや bot は使わなかった。

図 18 に測定結果を示す。TI は 30 秒である。横軸が最小経路の arc の本数であり、縦時間が結果が求まるまでの時間である。それぞれの arc の本数について、5 回ずつ測定を行っている。この図が示すように、 R^2 が 0.98 なので、計算時間はほぼ、arc の本数に比例している。その比例定数は、予測値である 15 に比較的近くなった。15 秒を超えている部分は計算時間が原因と思われる。

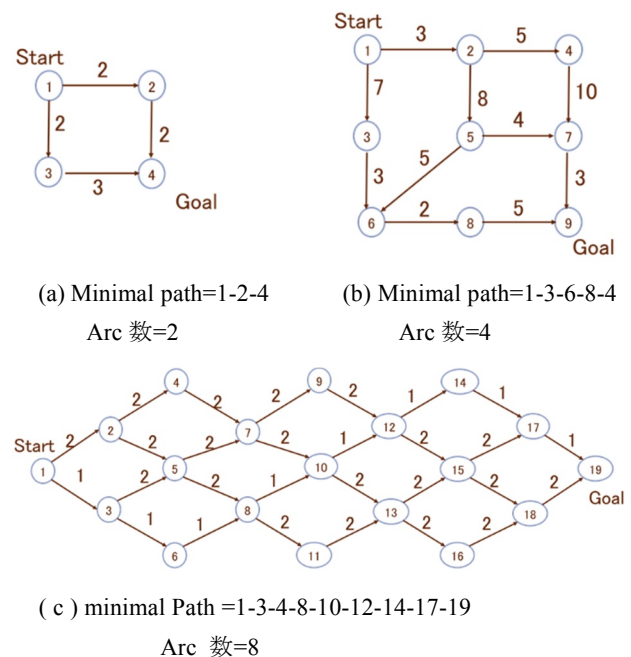


Figure 16. 最小経路を計算する時間の計測で使ったグラフ
Figure 14. Graphs for the measurement of the calculation time of minimal paths.



図 17. 測定の様子
Figure 15. Picture of the measurement.

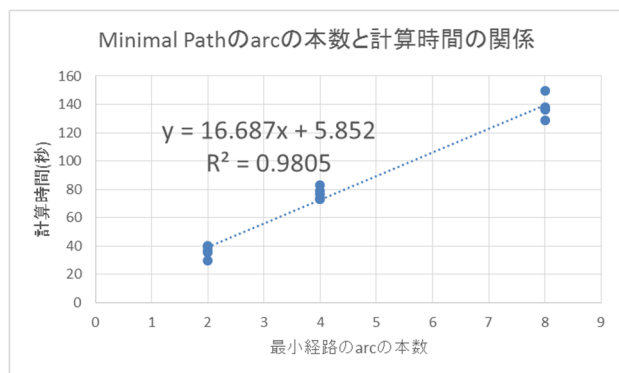


図 18. 測定結果より求めた最小経路の Arc の本数と計算時間の関係

Figure 16. Relationship between number of minimal path's arcs and measured times.

なお、切片が 0 であるはずなのに 5.85 秒となっている。これは計測する際に、Web ページを 10 秒間隔で更新していたため、実際の計算が終了してから平均しておよそ 5 秒経ってからしか結果を確認できなかったことが原因と思われる。

また、最小経路となる弧の数が、そうではない場合の弧の数より大きいとき、最初に goal node に現れる経路は最小経路ではない場合がある。しかしながら計算が繰り返されることにより、最終的には最小経路が求まり、それ以後は、goal node の結果はずっと最小経路とその値が示される。

6. 関連研究

6.1 VRRP

VRRP[14]はネットワーク機器の信頼性向上のため一般的に使われているプロトコルである。これに対して本論文で述べた信頼性向上の手法は、応用システムについて扱っている。

6.2 動的計画法の並列計算

動的計画法の並列計算については古くから[1][2]などの研究が行われている。これらの研究は、与えられた問題の解析や分解を行い、少ないCPUで大きな速度向上を図ることを主眼に置いている。これに対して、本論文で述べた並列計算は、CPUに相当する Bot や Wiki ページの資源は node 数に比例した数だけ利用できることを仮定しているが、とくに問題の解析や分解を行わなくても、最小経路の arc 長に比例した時間で計算が終了する。

6.3 SETI@Home

SETI@Home[15]は天文台の観測データの中に地球外知的生命体からの無線信号の証拠と見られるものがないか探索するプロジェクトである。ここで、探索には、一般から広く募られたボランティアの、インターネット接続されたパソコンのCPUの空き時間等が利用される。観測データは小さく切り分けられ、それらが数百万台のパーソナルコン

ピュータに配布され、分析され、分析結果が返される。通常なら最新のスーパーコンピュータを必要とするような分析をインターネット上のコミュニティの援助によって達成できる。

SETI@Home は一般から広く募られたボランティアの、大量の CPU を科学技術計算で利用できることを示した。今回の研究では、SETI@Home で行われる計算が固定的であるにに対して、汎用的な並列計算が Bot によって行えることを示した。

7. おわりに

Bot と Wiki を使った並列計算、「Wiki ページ型並列プログラミング」について述べた。Wiki ページ型並列プログラミングを使って、動的計画法による最小経路問題を解くことができた。このとき、今回の方法では、必要な計算時間は最小経路の弧の数に比例することを予測し、実測によりそれを確認した。今回の方法では、ボットの数は最初からノードの数に比例したものを用意することを前提としているため、CPU(ボット)の数の増加による、計算速度向上比は最初から考慮していない。ノードの数に比例した Bot や wiki ページなどの資源が必要になるため、ある意味、非常に効率が悪い方法である。しかしながら、SETI@Home が示したように、大量の Bot が使える可能性があるため、今回示した方法による最小経路計算も役に立つ可能性がある。

今回は failover 機能の検証は行っていない。セキュリティ強化についても、Basic 認証が使える以外にはなにも対策を行っていない。また、Class ページによって従来と比べてかなり記述量が減ったが、それでもまだ記述量が多い。今後これらのやり残したことを実施していく予定である。

謝辞 実験に協力してくれた学生諸君に感謝します。

参考文献

- [1] 瀬口 靖幸, 田中 正夫, 中島 利朗, 金田 悠紀夫, 小畑 正貴, 西野 佐登史, 動的計画法の並列計算 — 並列計算性とアルゴリズム —, 情報処理学会論文誌 Vol.26, No.5, (1985).
- [2] 程 翔, 相原玲二, 山下雅史, 阿江忠, 動的計画法に基づく資源配分並列アルゴリズム, 電子情報通信学会論文誌 D, 情報・システム 71(4), (1988).
- [3] Timothy Budd, An Introduction to Object-Oriented Programming, Pearson; 3 edition (October 22, 2001)
- [4] 山之上卓, 山本史弥, 小田謙太郎, 下園幸一, "Pukiwiki で Java プログラムの起動とそのデータ保存を可能とするシステムの試作", 情報処理学会 コンピュータと教育研究会第 109 回研究発表会 (CE 研究会), (2011 年 3 月)
- [5] 山之上 卓, 白澤竜馬, 小田謙太郎, 下園幸一, Wiki と携帯型遠隔操作端末を使った情報セキュリティ監視システム, 情報処理学会研究会報告, Vol. 2012-IOT-16, No.35, (2012).
- [6] Takashi Yamanoue, Kentaro Oda, Koichi Shimozono, A M2M system using Arduino, Android and Wiki Software, Proceedings of the 3rd IIAI International Conference on e-Services and Knowledge Management (IIAI ESKM 2012), pp.123-128, Fukuoka, Japan, 20-22 (Sep. 2012).

- [7] Takashi Yamanoue, Kentaro Oda, Koichi Shimozone. A Malicious Bot Capturing System using a Beneficial Bot and Wiki, Journal of Information Processing(JIP), vol.21, No.2, pp.237-245(2013).
- [8] Takashi Yamanoue, Kentaro Oda, Koichi Shimozone. An Inter-Wiki Page Data Processor for a M2M System, 4th International Conference on E-Service and Knowledge Management (ESKM 2013), Advanced Applied Informatics (IIAIAAI), 2013 IIAI International Conference on., pp.45-50, Matsue, Japan., (2013).
- [9] 山之上卓, Twitter と Wiki を使った自動情報提示システム, 情報処理学会, 研究報告インターネットと運用技術 (IOT) , vol. 2016-IOT-32, No.5, pp.1-7, (2016-03-03).
- [10] Blaise Barney, Introduction to Parallel Computing, https://computing.llnl.gov/tutorials/parallel_comp/
- [11] Twitter4J, <http://twitter4j.org/ja/>
- [12] Apache http components, <https://hc.apache.org>
- [13] <https://twitter.com/miyasita>
- [14] Virtual Router Redundancy Protocol (VRRP), IETF RFC 2338, 3768, 5798.
- [15] SETI@Home, <http://setiathome.ssl.berkeley.edu>