Martlet

A Scientific Work-Flow Language for Abstracted Parallisation

Daniel Goodman

Daniel.Goodman@ComLab.ox.ac.uk
Contents

- About climateprediction.net
- Example problem
- Martlet
- Extending use through JIT Compilers
- Conclusion
Introducing climateprediction.net

Use distributed computing to:
Introducing \texttt{climateprediction.net}

Use distributed computing to:

- improve the parametrisation of climate models both future and present
Introducing climateprediction.net

Use distributed computing to:

- improve the parametrisation of climate models both future and present
- generate a huge set of analysable perturbed climate model runs as a resource for scientists.
Introducing climateprediction.net

Use distributed computing to:

- improve the parametrisation of climate models both future and present
- generate a huge set of analysable perturbed climate model runs as a resource for scientists.

The framework to achieve this is currently built on top of the Berkeley Open Infrastructure for Network Computing (BOINC)
Achieved so far
Achieved so far

- Over 62,250 machines running models
Achived so far

- Over 62,250 machines running models
- Over 180,000 full climate models run
Achieved so far

- Over 62,250 machines running models
- Over 180,000 full climate models run
- Totalling over 17,100,000 years modelled
Map of Users

Users on all seven continents
Returned Runs

Returned runs create a data set that is:

- too big to return to a single location
- trivially partitioned
Returned Runs

Returned runs create a data set that is:

- to big to return to a single location
- trivially partitioned

Because of this we:

- ensure that individual runs are not split
- use many donated computers to store returned runs
Example problem

Locations of upload servers
Example problem

Upload servers holding recent data to query X
Example problem

\[ y_0 = \sum_{i=0}^{n_1-1} x_i \]
\[ z_0 = n_1 \]
\[ y_1 = \sum_{i=n_1}^{n_2-1} x_i \]
\[ z_1 = n_2 - n_1 \]
\[ \vdots \]
\[ y_{a-1} = \sum_{i=n_{a-1}}^{n_a-1} x_i \]
\[ z_{a-1} = n_a - n_{a-1} \]
\[ \bar{x} = \frac{\sum_{i=0}^{a-1} y_i}{\sum_{i=0}^{a-1} z_i} \]

Using existing workflow languages would require the user to know the value of \( a \) at the time they write the workflow, and to update the workflow when the value of \( a \) changes.
Returned Runs

As analysis is typically data intensive, this means most direct analysis of the data needs to occur locally to the upload servers. This requires that an interface is provided to:
Returned Runs

As analysis is typically data intensive, this means most direct analysis of the data needs to occur locally to the upload servers. This requires that an interface is provided to:

- Prevent users running their own binaries
Returned Runs

As analysis is typically data intensive, this means most direct analysis of the data needs to occur locally to the upload servers. This requires that an interface is provided to:

- Prevent users running their own binaries
- Hide the complexity of the distribution
Returned Runs

As analysis is typically data intensive, this means most direct analysis of the data needs to occur locally to the upload servers. This requires that an interface is provided to:

- Prevent users running their own binaries
- Hide the complexity of the distribution
- Hide the internal state from the user
A function for reversing a list

\[
\text{reverse} :: [a] \rightarrow [a] \\
\text{reverse} \; [] = [] \\
\text{reverse} \; (x:xs) = \text{reverse} \; xs \; ++ \; [x]
\]

Note the separation between the base case and the inductive case
define{
    uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){

    Y = new dismatrix(A);
    Z = new disinteger(A);
    ZTotal = new integer(B);

    map{
        matrixSum:uri1(A,Y);
        matrixCardinality:uri1(A,Z);
    }
}
Introducing Martlet

tree((YL,YR)\Y -> B, (ZL,ZR)\Z -> ZTotal){
    matrixSumToVector:uri1(YL,YR,B);
    integerSum:uri1(ZL,ZR,ZTotal);
}

matrixDivide:uri1(B,Z_Total,B);
}
Introducing Martlet

- Two classes of data structure
  - Local data
  - Distributed data
Introducing Martlet

- Two classes of data structure
  - Local data
  - Distributed data

- Two types of statement
  - Normal statements
  - Expandable statements
Introducing Martlet

- Two classes of data structure
  - Local data
  - Distributed data
- Two types of statement
  - Normal statements
  - Expandable statements
- Both functions and data structures are referenced by URIs
Introducing Martlet

- Two classes of data structure
  - Local data
  - Distributed data

- Two types of statement
  - Normal statements
  - Expandable statements

- Both functions and data structures are referenced by URIs

- Functions are first class values
Data Structures

define{
    uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){
    Y = new dismatrix(A);
    Z = new disinteger(A);
    ZTotal = new integer(B);

    map{
        matrixSum:uri1(A,Y);
        matrixCardinality:uri1(A,Z);
    }
}
Local Data Structures

define{
    uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B) {
    Y = new dismatrix(A);
    Z = new disinteger(A);
    ZTotal = new integer(B);

    map{
        matrixSum:uri1(A,Y);
        matrixCardinality:uri1(A,Z);
    }
}
Local Data Structures

Local data structure

- Exists only in a single location
- Can be acted on directly
Distributed Data Structures

define{
    uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){
    Y = new dismatrix(A);
    Z = new disinteger(A);
    ZTotal = new integer(B);

    map{
        matrixSum:uri1(A,Y);
        matrixCardinality:uri1(A,Z);
    }
}
Distributed data structure

- Abstractly, a list of local data structures
- Can be split across many locations
- Can be acted on only in conjunction with an expandable statement
Statements

Normal statements

- Sequential composition
- Asynchronous composition
- If-else
- While
- Temporary Variables
- Process calls
Statements

Expandable statements

- Map
- Foldr
- Foldl
- Tree
define{
    uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){
    Y = new dismatrix(A);
    Z = new disinteger(A);
    ZTotal = new integer(B);

    map{
        matrixSum:uri1(A,Y);
        matrixCardinality:uri1(A,Z);
    }
}
Map

Haskell definition

\[
\text{map } f \; [] = [] \\
\text{map } f \; (x:xs) = (f \; x):(\text{map } f \; xs)
\]
Map

Haskell definition

\[
\text{map } f \ [ ] = \ [ ] \\
\text{map } f \ (x:xs) = (f \ x) : (\text{map } f \ xs)
\]

Example Martlet code

map{
    f1(A);
    f2(A,B);
}
Map

The unexpanded abstract syntax tree

```
map    seq
   |    |
   |    |
f_1(A)  f_2(A, B)
```
The unexpanded abstract syntax tree

```
map
  seq
    f_1(A)
    f_2(A, B)
```

Concrete input data

\[ A = [A_1, A_2, A_3] \]
\[ B = [B_1, B_2, B_3] \]
Map

\[ \text{async} \rightarrow \text{seq} \]

\[ f_1(A_1) \]

\[ f_2(A_1, B_1) \]

\[ f_1(A_2) \]

\[ f_2(A_2, B_2) \]

\[ f_1(A_3) \]

\[ f_2(A_3, B_3) \]
tree((YL,YR)\Y -> B, (ZL,ZR)\Z -> ZTotal){
    matrixSumToVector: uril(YL,YR,B);
    integerSum: uril(ZL,ZR,ZTotal);
}

matrixDivide: uril(B,ZTotal,B);
}
Tree

Haskell definition

tree f [x] = x

tree f (x:y:ys) =
    f (tree f xs') (tree f ys')
    where (xs', ys') = split (x:y:ys)
Tree

Haskell definition

```haskell
tree f [x] = x
tree f (x:y:ys) =
    f (tree f xs') (tree f ys')
    where (xs', ys') = split (x:y:ys)
```

Example Martlet code

```martlet
tree((XL,XR)\X -> A){
    f(A,XL,XR);
}
```
Example Martlet code

tree((XL,XR) \ X \rightarrow A) {
    f(A,XL,XR);
}

The unexpanded abstract syntax tree

```
  tree
    f(A,XL,XR)
```
Tree

Example Martlet code

tree((XL,XR) \ X \rightarrow A) {
  f(A,XL,XR);
}

The unexpanded abstract syntax tree

Concrete input data

\[ X = [X_1, X_2, X_3, X_4, X_5] \]
Concrete input data
\[ X = [X_1, X_2, X_3, X_4, X_5] \]

Expanded concrete syntax tree
Tree

$f(A, A_1, X_5)$

$f(A_1, A_2, A_3)$

$f(A_2, X_1, X_2)$

$f(A_3, X_3, X_4)$

$X_1$ $X_2$

$X_3$ $X_4$
Example

\[ y_0 = \sum_{i=0}^{n_1-1} x_i \]
\[ z_0 = n_1 \]

\[ y_1 = \sum_{i=n_1}^{n_2-1} x_i \]
\[ z_1 = n_2 - n_1 \]

\[ \vdots \]

\[ y_{a-1} = \sum_{i=n_{a-1}}^{n_a-1} x_i \]
\[ z_{a-1} = n_a - n_{a-1} \]

\[ \bar{x} = \frac{\sum_{i=0}^{a-1} y_i}{\sum_{i=0}^{a-1} z_i} \]
Example

```c
define{
    uri1 = baseFunction:system:http://cpdn.net:8080/Martlet;
}

proc(A,B){

    Y = new dismatrix(A);
    Z = new disinteger(A);
    ZTotal = new integer(B);

    map{
        matrixSum:uri1(A,Y);
        matrixCardinality:uri1(A,Z);
    }
}
```
Example

tree((YL,YR)\Y -> B, (ZL,ZR)\Z -> ZTotal){
    matrixSumToVector:uri1(YL,YR,B);
    integerSum:uri1(ZL,ZR,ZTotal);
}

matrixDivide:uri1(B,Z_Total,B);
}
Example

- create Y, Z, ZTotal
- seq
- map
- MatrixSum(A, Y)
- MatrixCardinality(A, Z)
- seq
- MatrixSumToVector(YL, YR, B)
- tree
- seq
- MatrixDivide(B, ZTotal, B)
- seq
- IntegerSum(ZL, ZR, ZTotal)
JIT Compliers

- Produce XML to represent the concrete syntax tree
- Use a JIT compiler to convert this to the workflow language of your choice
- Execute on existing resources
- Store returned results
Conclusions
Conclusions

- Solution to a new style of problem
Conclusions

- Solution to a new style of problem
- Can be used to build on existing resources with relative ease and without affecting their existing functionality
Conclusions

- Solution to a new style of problem
- Can be used to build on existing resources with relative ease and without affecting their existing functionality
- Ideas can be absorbed into the next generation of workflow languages
Conclusions

- Solution to a new style of problem
- Can be used to build on existing resources with relative ease and without affecting their existing functionality
- Ideas can be absorbed into the next generation of workflow languages

More information please email Daniel.Goodman@comlab.ox.ac.uk