Scalable Multi-Language Data Analysis on BEAM
The Erlang Experience

Jörgen Brandt

Humboldt-Universität zu Berlin

2016-09-08

Erlang User Conference 2016
About me

- PhD student at Humboldt (Berlin)
- Creator of Cuneiform workflow language
- Major area of application: Bioinformatics
Motivation

- Introduction to Cuneiform
- How to implement a large-scale data analysis PL
- How in Erlang?
- Why in Erlang?
Scientific Workflow Systems
Workflows as DAGs

- Scientific Workflows are DAGs
Scientific Workflow Systems

Workflows as DAGs

- Scientific Workflows are DAGs
- Nodes are tasks
Scientific Workflow Systems

Workflows as DAGs

- Scientific Workflows are DAGs
- Nodes are tasks
- Edges are data dependencies
The Next Generation Sequencing use case

Reference alignment:

Variant calling:

Variant characterization:
The Next Generation Sequencing use case

Jörgen Brandt (HU Berlin)
The Next Generation Sequencing use case

Jörgen Brandt (HU Berlin)
The Next Generation Sequencing use case

Jørgen Brandt (HU Berlin)

The Erlang Experience

2016-09-08 6 / 29
Cuneiform is ...

Functional Language for Large-Scale Data Analysis
implemented in Erlang

- Different from systems like Spark
  - Standalone syntax, no fluent API in Scala
  - Integration of foreign PLs

- Different from distributed workflow languages like Snakemake
  - Reduction of functional expression, no static dependency graph
The main idea

Functional Programming
+
Foreign Language Interfacing
Functional Programming

- Very expressive
- Natural operations on lists (map, ...)
- Natural iteration (recursion)

Gain

- General data analysis
Functional Programming

- Very expressive
- Natural operations on lists (map, ...)
- Natural iteration (recursion)

Gain
- General data analysis

Gain
- Parallelize independent sub-expressions
- Lazy Evaluation

Gain
- Automatic Parallelism
Cuneiform Example

deftask gunzip( out( File ) : gz( File ) ) in bash *{
  out=output.file
  gzip -c -d $gz > $out
}
deftask gunzip( out( File ) : gz( File )) in bash *
{
  out=output.file
  gzip -c -d $gz > $out
}*

gunzip(
  gz: 'myarchive1.gz'
);

Cuneiform Example
deftask gunzip(out(File) : gz(File)) in bash *
  out=output.file
  gzip -c -d $gz > $out
}*

gunzip(
  gz: 'myarchive1.gz' 'myarchive2.gz'
);
Cuneiform Example

```bash
deftask gunzip( out( File ) : gz( File ) ) in bash *
    out=output.file
    gzip -c -d $gz > $out
*

gunzip(
    gz: 'myarchive1.gz' 'myarchive2.gz' 'myarchive3.gz'
);
```
Available example workflows:

- Variant Calling (Varscan)
- Methylation
- RNA-Seq
- Variant Calling (GATK)
- etc
Example: RNA-Seq

cuneiform-lang.org/examples

RNA-Seq
Feb 26, 2016

This workflow exemplifies the comparison of DNA expression levels in two experimental conditions from RNA-Seq data. It reimplements a study by Trapnell et al. 2012.

Figure 1: Probability density of expression levels of transcribed genes in group C1 (blue) and group C2.

<table>
<thead>
<tr>
<th>Cookbook</th>
<th><a href="https://github.com/joergen7/rna-seq">https://github.com/joergen7/rna-seq</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>Cuneiform script</td>
</tr>
<tr>
<td>Publication</td>
<td>Trapnell et al. 2012</td>
</tr>
<tr>
<td>Input data</td>
<td>Drosophila reference genome SAMtools reference index Bowtie Index FastQ samples (2 conditions)</td>
</tr>
</tbody>
</table>
Flow-based programming and Erlang style message passing - A Biology-inspired idea of how they fit together

Posted on: 13 Jun '15
I think Erlang/Elixir fits great as control plane or service-to-service messaging layer for distributing services built with flow-based programming, and in this post, I'll tell you why.

Edit I: Fixed misleading heading: FBP at all suitable for distributed computing? -> Further differences between the paradigms] (June 15, 17:07 CET)
Edit II: FYI: Interesting discussions on the post is happening on HackerNews, and the Flow-based programming mailing list, Erlang VM very interesting - so is Flow-based programming
Edit III: Typo corrections
Edit IV: FYI: Elixir/FPB creator Peter C Marks blogged a comment of this post and the discussion it triggered Check it out exvifp.org! - Let the discussion and exploration continue! :) 
Note V: The slowness of Erlang/Elixir that I mention, is disputed! See discussion on this meetup page, and in particular Johan Lind's improvements on the code example here
Biology-Inspired Analogy

- Large-scale data analysis systems as 2-tier system:
  - Algorithm-level (fast, local)
  - Workflow-level (organizational, distributed)
- Analogy to the cell:
  - DNA transcription (fast, local)
  - Cell-to-cell signaling (organizational, distributed)

Erlang good fit for organizational part
Distributed Workflow System Architecture

Query

Query

Query
Distributed Workflow System Architecture

Query → Cache → Query

Query

Query

Query
Distributed Workflow System Architecture
Distributed Workflow System Architecture
Two Modeling Challenges (i)

- Interpreter
  - Reduction of query expression

- Execution Environment
  - Distributed System
How to model programming languages?
Parser is generated from BNF

Interpreter is transcribed from Operational Semantics
Abstract Syntax

An expression in Cuneiform is . . .

- Expressions can contain other expressions
- Semantics define how expressions are reduced

\[ x_0 \rightarrow x_1 \rightarrow \ldots \rightarrow x^* \]
Implementing an Operational Semantics in Erlang

4.1 Expression

The first of two items constituting Cuneiform abstract program is the query expression list \( \mathbf{E} \in \mathbb{E} \) where \( \mathbb{E} \) is the set of lists whose elements are expressions. An expression \( e \in \mathbb{E} \) in turn, is either a string literal, a variable, a future channel selection, a conditional, or a task application:

\[
\mathbf{E} = \mathbf{S} \cup \mathbf{V} \cup \mathbf{S}_{\text{C}} \cup \mathbf{S}_{\text{D}} \cup \mathbf{A}
\]

(1)

We introduce each of these syntactical items in turn. Figure 3 summarizes the abstract syntax of Cuneiform expressions.

4.1.1 String Literal

The string literal is Cuneiform’s only basic data type. In Cuneiform’s abstract syntax, a string literal is represented as an abstract term of type \( \mathbf{S} \), i.e., a 2-tuple whose first element is the symbol \( \mathbf{S} \). The second element of the tuple is a character sequence \( r \) holding the context of the string literal. Thus, let the set \( \mathbf{S} \) denote the set of strings (quoted character sequences), then the set \( \mathbf{S} \) of string literals is given by the rule:

\[
x \in \mathbf{S} \\
(\text{str},x) \in \mathbf{S}
\]

(2)

4.1.2 Variable

Variables are placeholders for expression lists. In the abstract syntax, a variable is represented by an abstract term of type \( \mathbf{V} \). It comprises the symbol \( \mathbf{V} \) and a character sequence holding the variable’s identifier. Thus, let \( \mathbf{V} \) denote the set of strings, then the set \( \mathbf{V} \) is given by the rule:

\[
x \in \mathbf{V} \\
(\text{var},x) \in \mathbf{V}
\]

(3)
Two Modeling Challenges (ii)

- Interpreter
- Execution Environment

Interpreter

- Reduction of query expression

Execution Environment

- Distributed System
How to model distributed systems?
How to Model Distributed Systems

Petri Nets

- Mature theory
  - Liveness
  - Invariants
  - Traps/Cotrails
  - ...
- Local decision/synchronization
- Parallel execution of independent transitions
Modeling the Cuneiform Execution Environment

\begin{align*}
a \in \text{App}, & \quad p \in \text{Pid}, \quad r \in \text{Result} \cup \text{Err}, \\
q \in \text{Req}, & \quad s \in \text{Spec} 
\end{align*}

\begin{enumerate}
\item \text{submit} \quad (a, q) \\
\item \text{resubmit} \quad (p, s, a, q) \\
\item \text{reply} \quad (p, s, a, q) \\
\item \text{schedule} \quad p_{\text{match}}(s, q) \\
\item \text{decommission} \\
\item \text{monitor}
\end{enumerate}
A generic Petri net OTP library

Copyright © 2016 Jørgen Brandt

Version: 0.1.0

Authors: Jørgen Brandt (brandje@hu-berlin.de).

References
  - Source code hosted at GitHub.

See also: cvm2, gen_pnet.

Some applications exhibit behavioral patterns that lend themselves to Petri Nets. The major advantage of modeling applications with Petri Nets is that they provide a natural view on the concurrent behavior of an application. This is achieved by making explicit the preconditions for an operation to be carried out while leaving implicit how and when an operation is triggered and what other operations might run in parallel.

This OTP library is a framework for programming with Petri Nets. It implements a very general form of Petri Nets: Colored Petri Nets (CPN). I.e., tokens may not only be markers but can be any conceivable data structure. Furthermore, a place can hold any number of tokens not just one.

While many simulation libraries only mimic the concurrent behavior of Petri Nets, the gen_pnet library allows the definition of Nets with an arbitrary number of transitions competing for a place’s tokens neither imposing order in the form of an overarching loop nor otherwise constraining parallelism.

Quick Start

This Quick Start section provides an overview about how Petri nets are started, queried, and manipulated with the gen_pnet module. We demonstrate the module’s API in terms of a cookie vending machine implemented in the cvm2 module which is also part of this code repository. Then, we have a look at how the callback functions of the cookie vending machine are implemented.
Flow-based programming revisited

- System languages for heavy lifting
- Large-scale data analysis is hard
  - Because programming languages are hard
  - Because distributed systems are hard
- Erlang is good fit
  - Because FP is already close to operational semantics style notation
  - Because Erlang process model already close to Petri Nets
cuneiform-lang.org
Conclusion

- Functional Programming + Foreign Languages
- Distributed Execution
  - Local Multicore
  - HTCondor
  - Hadoop
- Automatic Parallelization
- Expressive data analysis workflows
- Foreign Language Interface
  - Bash
  - Perl
  - ...

Cuneiform
Distributed Functional Programming with Foreign Language Interfacing

cuneiform-lang.org
Questions?