Scaling Erlang to 10,000 cores

Simon Thompson, University of Kent
Multicore and many-core

The inexorable rise in core numbers …

… growing exponentially just as processors used to.

These are becoming the standard platforms for general-purpose systems.
Languages and tools

What are the right programming models and tools …

… for building general-purpose software on these platforms?
Requirements

Robust against core failure …

… scalable now and in the future.
The aim of RELEASE

To scale the actor, concurrency-oriented, paradigm …

… to build reliable general-purpose software, such as server-based systems, …

… on massively parallel machines (10^5 cores).
Build on Erlang!

Erlang/OTP has inherently scalable computation and reliability models.
Multicore Erlang
Distribution and core failure
Distribution and core failure
Design choices

Erlang multicore is “black box” …

… we don’t change that

… but we do need to observe behaviour at that level.

Current Erlang implementation: core failure $\rightarrow$ host failure …

… future technology may change that

… our focus is on scaling host numbers
Build on Erlang?

Scalability is constrained in practice …

… VM aspects: synchronisation on internal data structures …

… language aspects, e.g. fully connected network of nodes, explicit process placement …

… tool support.
Building on Erlang/OTP

Scalable Infrastructure

Scalable Distributed Erlang

The Virtual Machine

Tools

Case Studies
The Virtual Machine
“Are we there yet?”

BenchErl is a publicly available scalability benchmark suite for applications written in Erlang. In contrast to other benchmark suites, which are usually designed to report a particular performance point, our benchmark suite aims to assess scalability, i.e., a set of performance points that show how an application’s performance changes when additional resources (e.g. CPU cores, schedulers, etc.) are added.

**MOTIVATION**

The concurrency model of Erlang is one of its most advertised features. However, understanding the behaviour of a highly concurrent Erlang application and most importantly detecting the bottlenecks that hinder the exploitation of a large number of CPU cores has not been an easy task. A tool that would help towards this direction has been missing for Erlang.

The features included in BenchErl allow the execution of applications in various execution environments, the visualization of the results, and the extraction of useful conclusions. Hence, it is a tool that might help the Erlang community make a first step to better understand the parameters that affect the parallel execution of Erlang applications.

**KEY FEATURES**

- **Unique**: BenchErl is the only benchmark suite that targets the scalability of Erlang applications.
- **Configurable**: BenchErl allows the configuration of a large number of parameters that might affect the execution of a benchmark. The execution of the benchmark with all possible combinations of these parameters is handled by BenchErl.
- **Automated**: BenchErl handles the collection, the execution and the visual presentation of the benchmark execution results.
- **Extendable**: It is straightforward to add new benchmarks and applications to BenchErl.
Improved VM infrastructure

Evolutionary changes in ETS storage … and proposals for more.

Memory allocation / deallocation … less locking … more scalable.

Better organisation of process and port tables … less locking needed.

More scalable internal management of processes / port signals … …avoiding heavy contention when much incoming + outgoing data.

Non-blocking mechanisms for loading code and setting tracing support.

Algorithm preserving term sharing in copying and message passing … … and its low-level implementation on the Erlang VM.

Already in R16 … except the last.
Figure 6. Scalability of ETS tables of type set across Erlang/OTP releases using a workload with 99% lookups and 1% updates.
Figure 7. Scalability of ETS tables of type `ordered_set` across OTP releases using a workload with 99% lookups and 1% updates.
Figure 11. Scalability of ETS on a workload with 99% lookups and 1% updates when varying the ETS table concurrency options.
Scaling ETS - lessons learned

- `ordered_set` needs to be fixed or replaced
- Locking is (still) a problem, but got better
- NUMA is a problem
- Reader groups may be not that important

Some general advice

- Use pinning on NUMA
- Use `read_concurrency` when doing only lookups
- Use `write_concurrency`
- Measure your use case when combining them
Eating our own dog food …

Applied the techniques of the project to our own systems …

… Dialyzer, and …

… Wrangler.
SD Erlang
Scalable distribution: SD Erlang

Patterns for interconnection.

Semi-explicit process deployment.
Distribution “out of the box”

Completely connected: all nodes connected to each other.

Quadratic complexity.
Scalability
Scalability

http://www.dcs.gla.ac.uk/~amirg/publications/ScalablePersistentStorage.pdf
Distribution “out of the box”

Completely connected: all nodes connected to each other.

Quadratic complexity.
SD Erlang “out of the box”

Complete connectivity within each s_group.

Overlap topology supports nesting, hierarchy and ad hoc models.
Speedup
Scalability
s_group operations

Create and delete s_groups.

Add and remove nodes from an s_group.

Return information about s_groups and their contents.

Register, re-register and unregister names in an s_group.

Send a message to a named process.

Information about names and whereabouts of named processes.

Based on the implementation of global groups in Erlang/OTP.
Semi-explicit placement

`s_group:choose_nodes([\{s_group, SGroupName\}] )`

Choose eligible nodes for spawn from the identified `s_group`.

`s_group:choose_nodes([\{attribute, AttributeName\}] )`

Choose eligible nodes which have the given attribute.

Attributes include proximity, load, … .
Getting it right

```eqc:quickcheck(prop_s_group()).
```

We built an executable operational semantics to model our implementation.

We used property-based testing with a state machine to check compliance between the semantics and the implementation.

Two errors in the semantic specification.

Two errors in the s_group implementation.

Two inconsistencies between the two.
Scalable Infrastructure
WombatOAM is an operations and maintenance framework for Erlang based systems.

It gives you full visibility on what is going on in Erlang clusters …

… either as a stand-alone product or by integrating into existing OAM infrastructure.
How it looks
WombatOAM

Monitor managed nodes liveliness

Group nodes by Erlang releases

Deploy Erlang releases in the cloud

Gather metrics from different sources, show them in graphs

Capture logs, show error and crash logs promptly

Show alarms raised by different applications in managed nodes
# Alarms in WombatOAM

<table>
<thead>
<tr>
<th>State</th>
<th>Severity</th>
<th>Date</th>
<th>Source</th>
<th>Alarm id</th>
<th>All:</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleared</td>
<td>major</td>
<td>09:24:34</td>
<td>cluster-node2@10.100.0.132</td>
<td>node_down</td>
<td>✔</td>
</tr>
<tr>
<td>cleared</td>
<td>major</td>
<td>09:24:34</td>
<td>cluster-node1@10.100.0.132</td>
<td>node_down</td>
<td>✔</td>
</tr>
<tr>
<td>cleared</td>
<td>major</td>
<td>09:24:34</td>
<td>cluster-node3@10.100.0.132</td>
<td>node_down</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>minor</td>
<td>2014-05-26 14:08:16</td>
<td>cluster-node3@10.100.0.132</td>
<td>old_code_minor</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>major</td>
<td>2014-05-26 14:08:16</td>
<td>cluster-node3@10.100.0.132</td>
<td>disk_capacity_major</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>indeterminate</td>
<td>2014-05-26 14:08:16</td>
<td>cluster-node3@10.100.0.132</td>
<td>{disk_almost_full,&quot;/boot&quot;}</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>minor</td>
<td>2014-05-26 14:08:15</td>
<td>cluster-node2@10.100.0.132</td>
<td>old_code_minor</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>major</td>
<td>2014-05-26 14:08:15</td>
<td>cluster-node2@10.100.0.132</td>
<td>disk_capacity_major</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>indeterminate</td>
<td>2014-05-26 14:08:15</td>
<td>cluster-node2@10.100.0.132</td>
<td>{disk_almost_full,&quot;/boot&quot;}</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>minor</td>
<td>2014-05-26 14:08:15</td>
<td>cluster-node1@10.100.0.132</td>
<td>old_code_minor</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>major</td>
<td>2014-05-26 14:08:15</td>
<td>cluster-node1@10.100.0.132</td>
<td>disk_capacity_major</td>
<td>✔</td>
</tr>
<tr>
<td>new</td>
<td>indeterminate</td>
<td>2014-05-26 14:08:15</td>
<td>cluster-node1@10.100.0.132</td>
<td>{disk_almost_full,&quot;/boot&quot;}</td>
<td>✔</td>
</tr>
</tbody>
</table>
Tools
Wrangler

Refactoring infrastructure

API: to write new refactorings from scratch

DSL: for “scripting” refactorings, supporting scaling

Introducing s_groups, and other parallel constructs.

Groups to s_groups.

Dog food: we’ve parallelised Wrangler, too.
Concuerror

“Debugging race conditions in concurrent programs is sometimes a sad story.”

- Stavros Aronis

Explores all interleavings of the processes, focusing on pairs of "racing" events …

… if a process crashes, Concuerror will then give you a detailed log of the events that lead up to the crash.

Case studies for Mochiweb and Poolboy.

http://concuerror.com
Profile … analyse … display in a browser, enhancing Percept.

Percept: active processes vs. time, drill down to process info … … including runnability, start/end time, parent/child processes, etc.

Enhancements: scheduler info, process communication, run-queue migration, runnable vs running, dynamic callgraph, links to source code, distribution support, etc.

Scalability: scalable process tree, selective profiling, parallel analysis and caching history webpages.

https://github.com/RefactoringTools/percept2
Improving Wrangler using Percept2
Improving Wrangler using Percept2

```haskell
[generalise_and_hash_file_ast_1(
    File, Threshold, ASTPId, true, SearchPaths, TabWidth)
  || File <- Files]
```

```haskell
para_lib:pmap(
    fun(File) -> generalise_and_hash_file_ast_1(
        File, Threshold, ASTPId, true, SearchPaths, TabWidth)
    end, Files)
```
Improving Wrangler using Percept2

```lisp
(examine_clone_candidates([], Thresholds, CloneCheckerPid, _Num) ->
  get_final_clone_classes(CloneCheckerPid);
(examine_clone_candidates([C|Cs], Thresholds, CloneCheckerPid, Num) ->
  output_progress_msg(Num),
  NewClones = examine_a_clone_candidate(C, Thresholds),
  add_new_clones(CloneCheckerPid, {C, NewClones}),
  examine_clone_candidates(Cs, Thresholds, CloneCheckerPid, Num+1).

(examine_clone_candidates(Cs, Thresholds, CloneCheckerPid) ->
  NumberedCs = lists:zip(Cs, lists:seq(1, length(Cs))),
  para_lib:pforeach(fun({C, Nth}) ->
    examine_a_clone_candidate(
      {C, Nth}, Thresholds, CloneCheckerPid)
    end, NumberedCs),
  get_final_clone_classes(CloneCheckerPid).
(examine_a_clone_candidate({C, Nth}, Thresholds, CloneCheckerPid) ->
  output_progress_msg(Nth),
  NewClones = examine_a_clone_candidate(C, Thresholds),
  add_new_clones(CloneCheckerPid, {C, NewClones}).
```

Devo

https://github.com/RefactoringTools/devo
Tracing Erlang

Enhancements to Erlang tracing … augmenting the VM

Logging only inter-node messages.

Filtering log messages.

DTrace/SystemTap support

Added probes.

Back-end for Percept2
Case studies
Discrete simulation engine aiming for maximum concurrency … … with both parallel and distributed modes of operation.

It focuses notably on scalability, in order to handle simulation cases which may be very large (potentially involving millions of interacting instances of models).

http://researchers.edf.com/software/sim-diasca-80704.html
Port Erlang to IBM BlueGene/Q
Summing up
University of Glasgow, University of Kent, Uppsala University and ICCS, National Technical University of Athens.

Erlang Solutions Ltd, Ericsson AB, Electricité de France.

October 2011 – February 2015

The project partners acknowledge the support of the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 287510.

www.release-project.eu
Questions?