RELEASE

Scaling Erlang to 10,000 cores

Simon Thompson, University of Kent

Erlang SOLUTIONS

UPPSALA UNIVERSITET

HERIOT WATT UNIVERSITY

eDF

University of Kent

ΕΠΙΣΕΥ ICCS

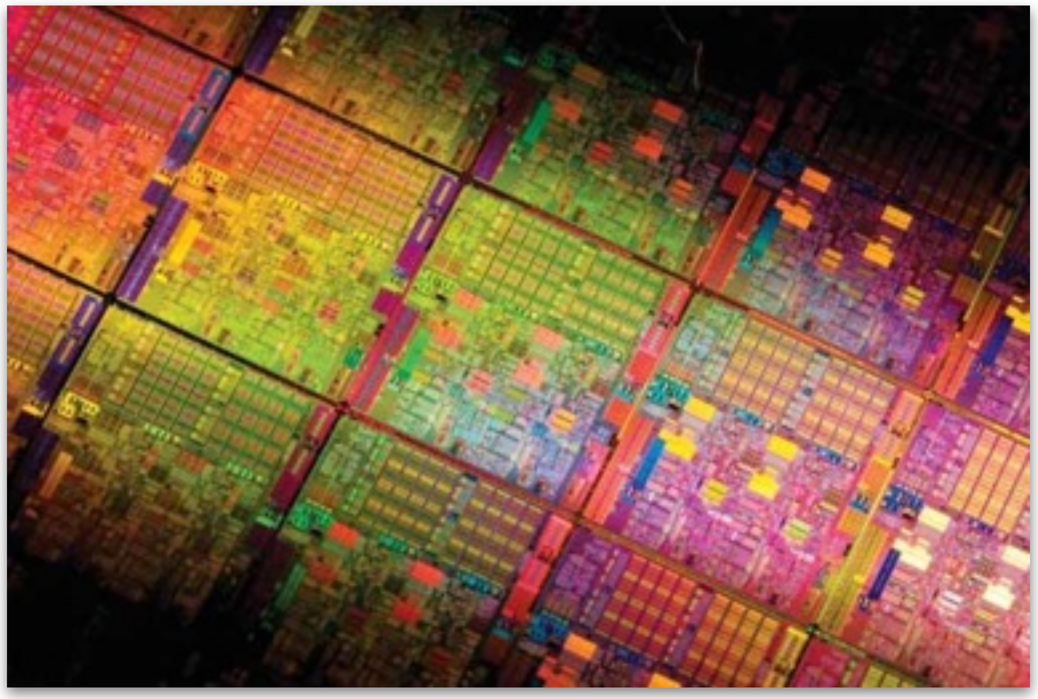
University of Glasgow

ERICSSON

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, …

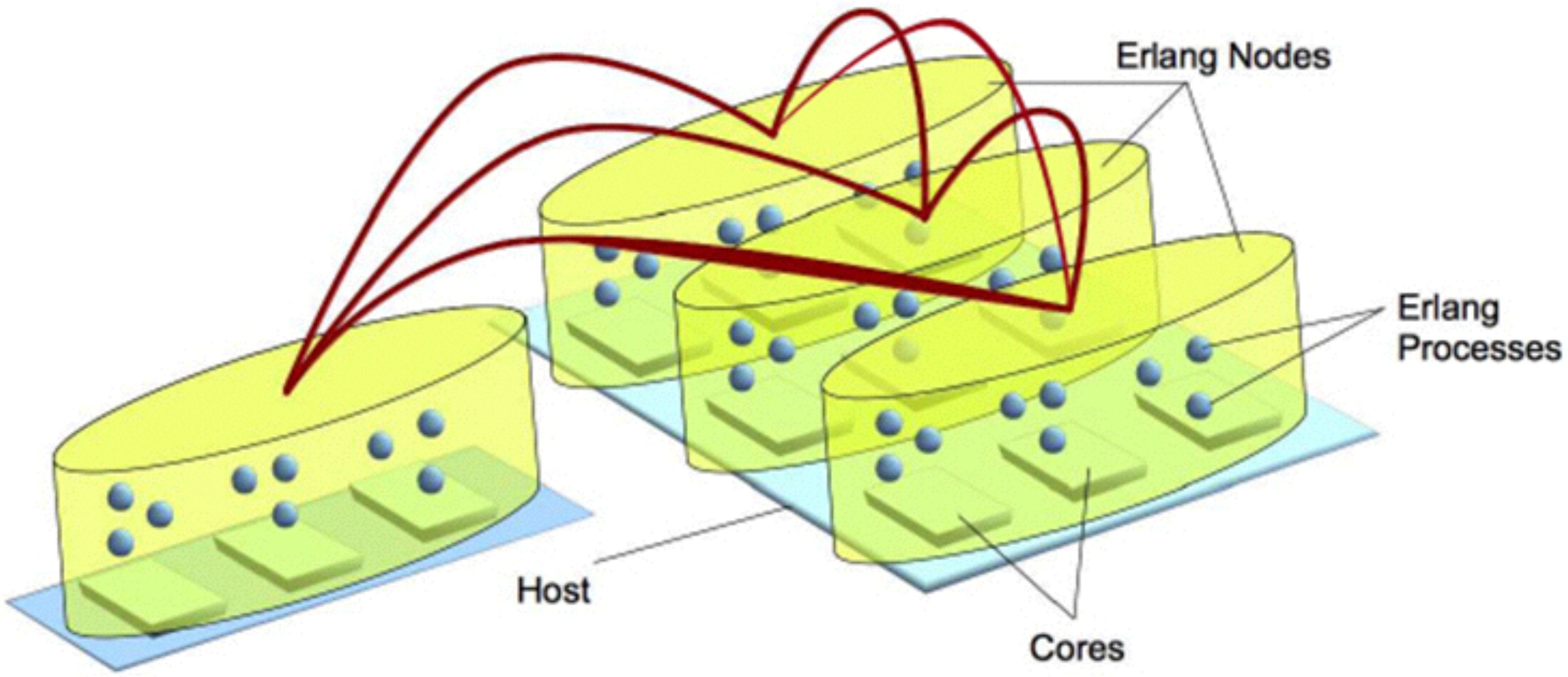
… on massively parallel machines (10^5 cores).

Build on Erlang!

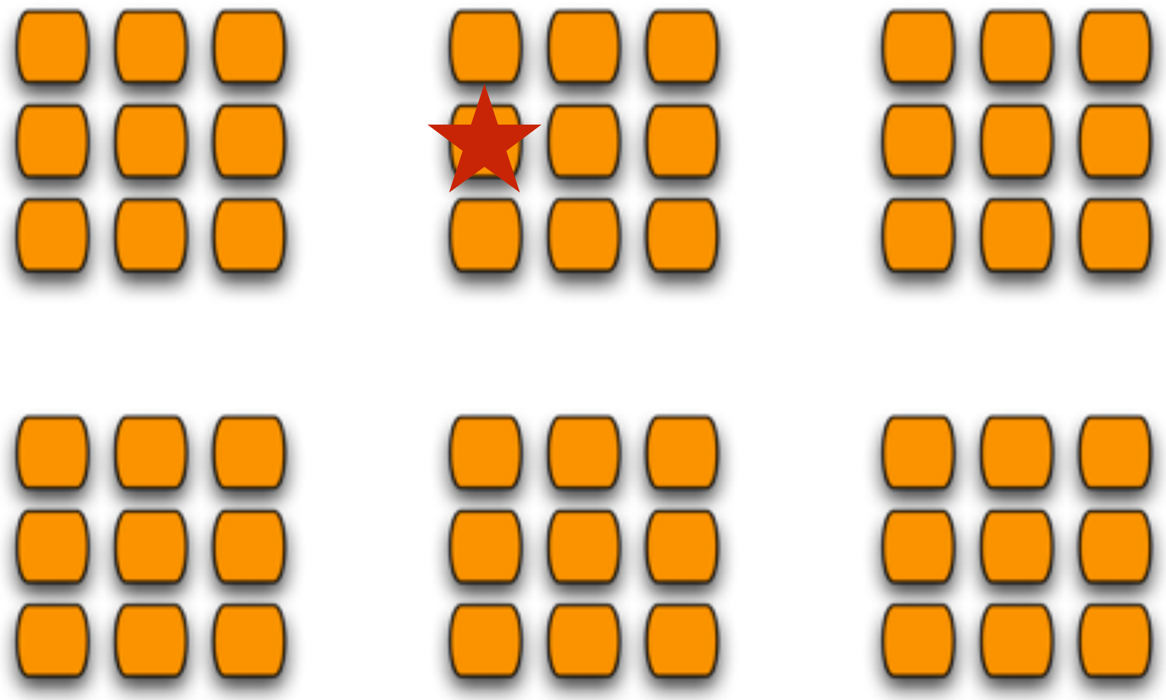
Erlang/OTP has inherently scalable computation and reliability models.

FACTORY
Erlang
Explicit parallelism
Sequential
pthread, CILK, OpenMP
Erlang, Parallell Haskell, skel (patterns)
C
SML, OCaml
Imperative
Declarative
It's weird.
It's simple.
It's extremely productive
It's extremely reliable
The design of the VM is beautiful
It's kinda slow
Erlang The Movie
Leverage Your Programming Skills with Erlang!
Learn You Some Erlang for Great Good!
Erlang

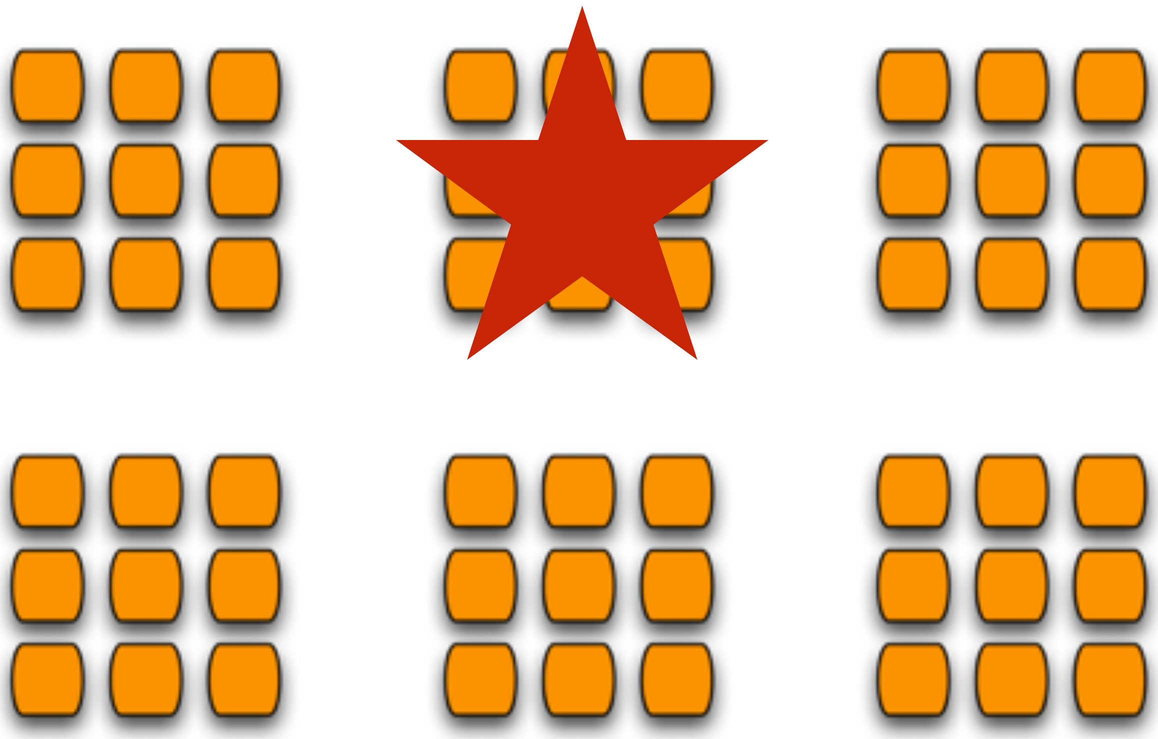
Multicore Erlang



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 → 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 …

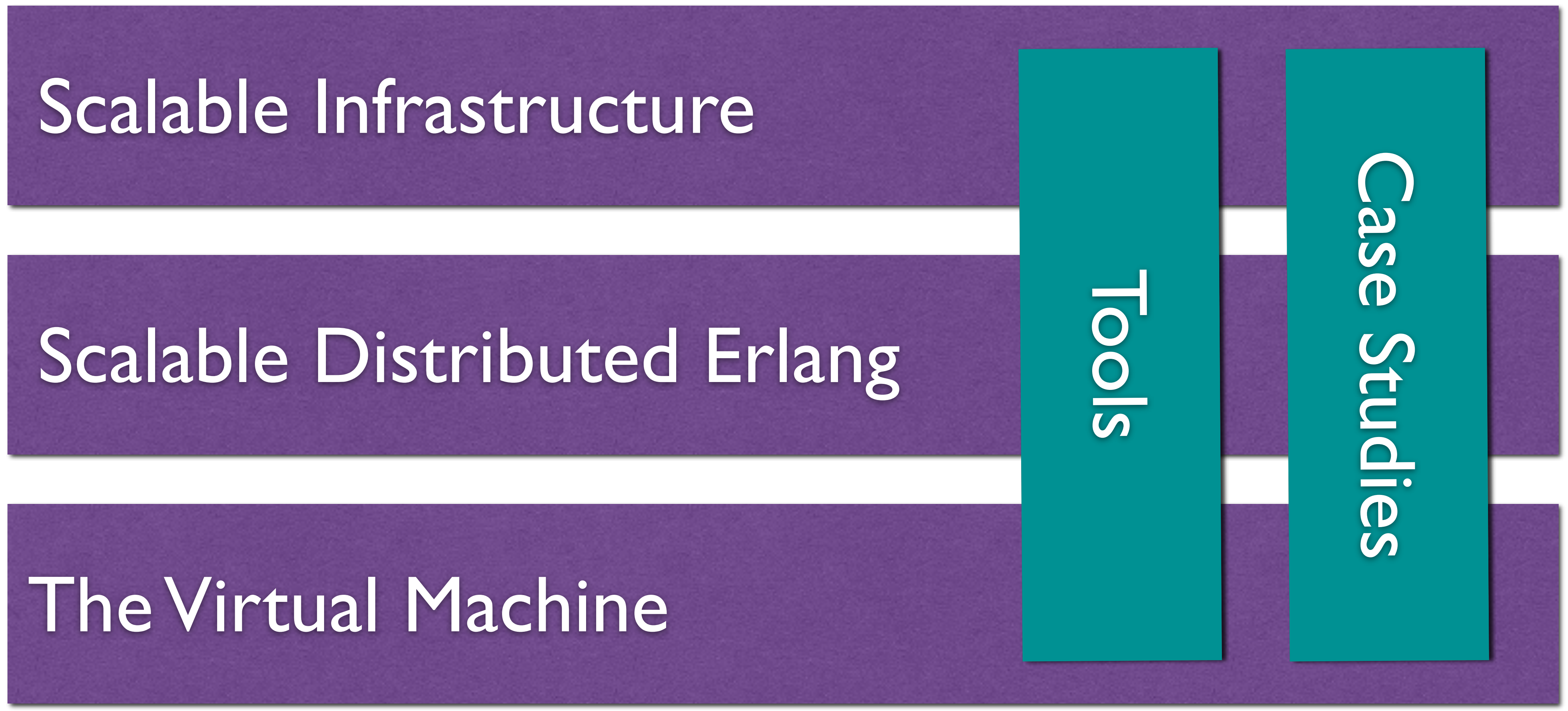
… language aspects, e.g. fully connected network of nodes, explicit process placement …

… tool support.

FACTORY
Explicit parallelism
Sequential
pthread, CILK, OpenMP
Erlang, Parallell Haskell, skel (patterns)
C
SML, OCaml
Imperative
Declarative
Leverage Your Programming Skills with Erlang!
Learn You Some Erlang for Great Good!
Erlang

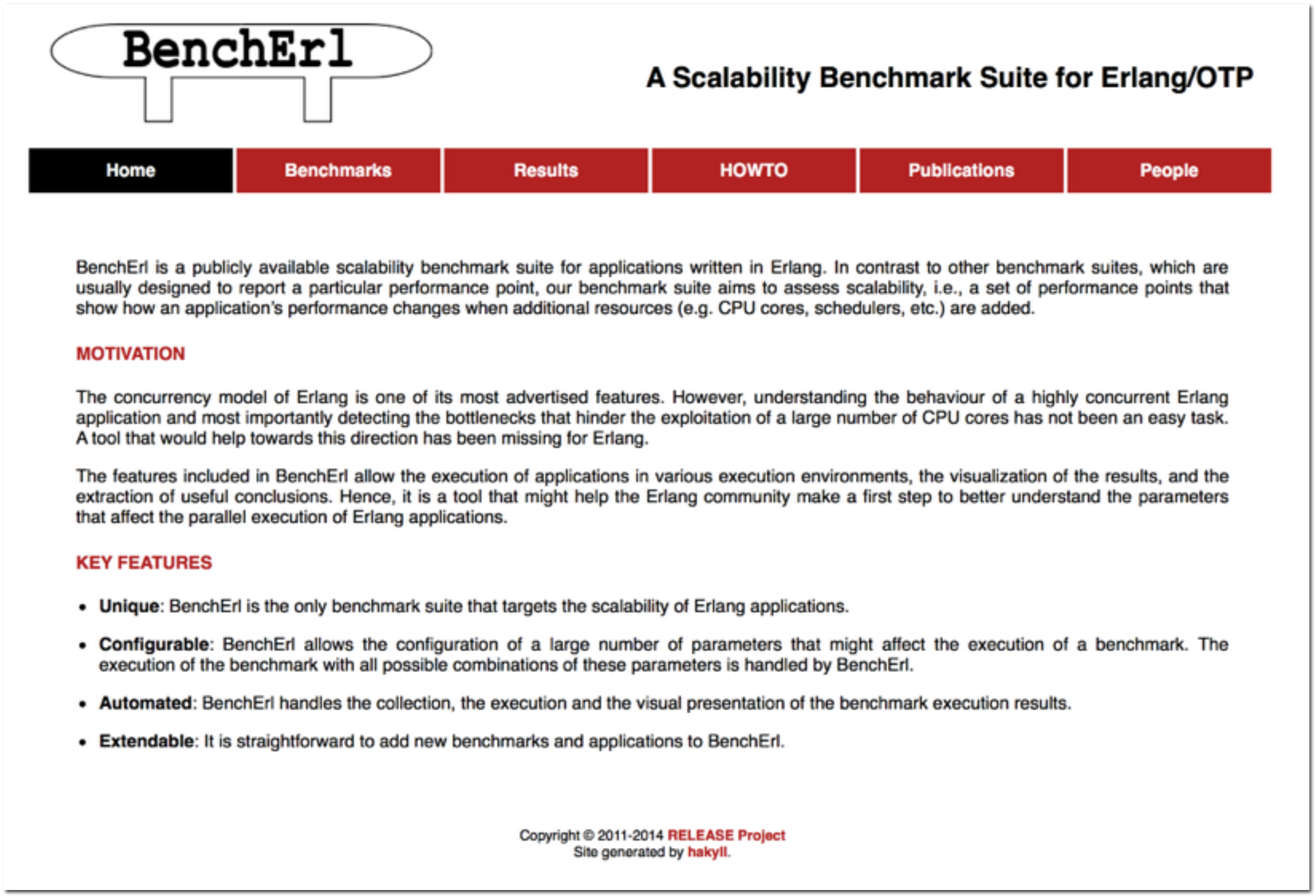
- It's weird.
- It's simple.
- It's extremely productive
- It's extremely reliable
- The design of the VM is beautiful
- It's kinda slow

Building on Erlang/OTP



The Virtual Machine

"Are we there yet?"



A Scalability Benchmark Suite for Erlang/OTP

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

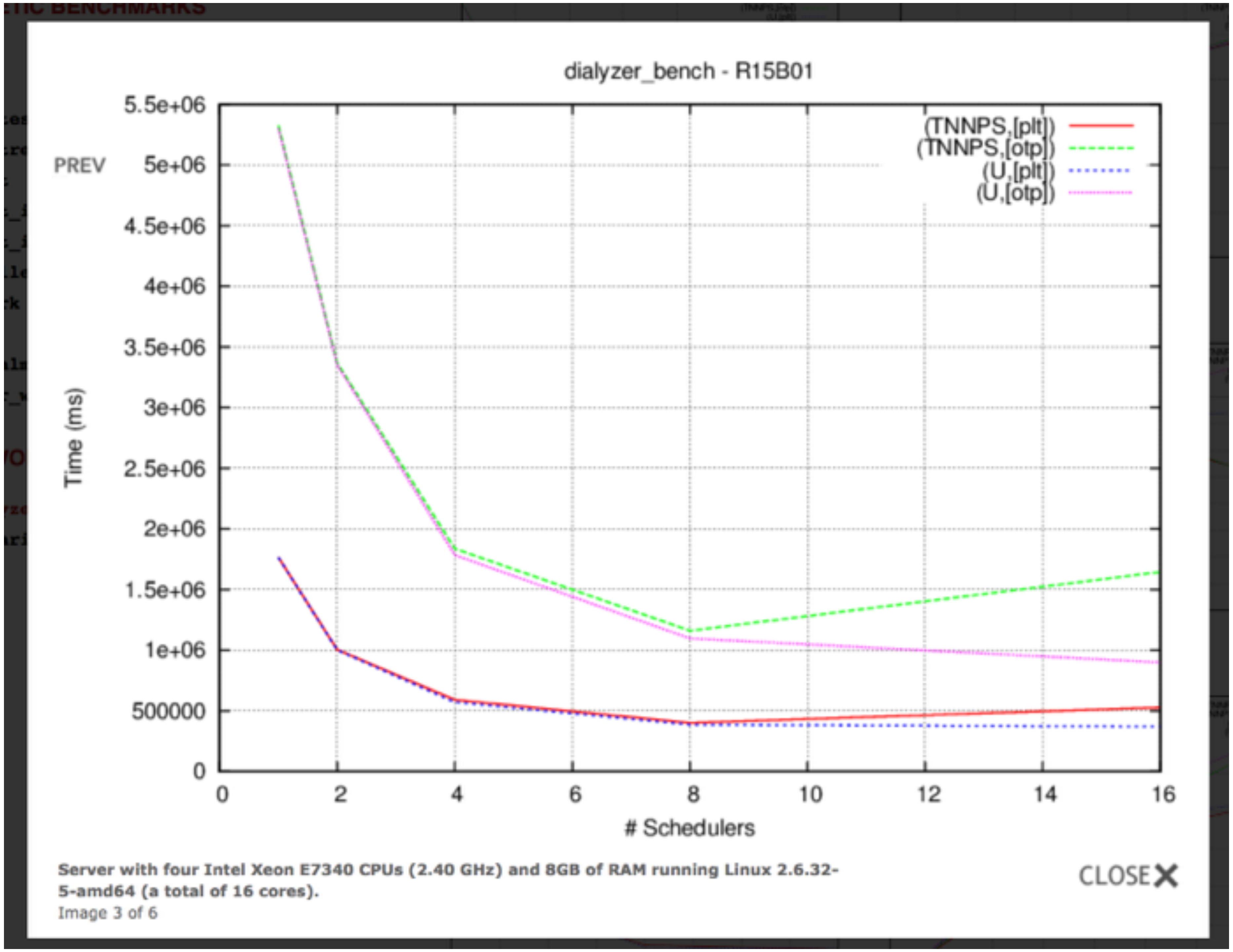
The concurrency model of Erlang is one of its most advertised features. However, understanding the behaviour of a highly concurrent Erlang application and most importantly detecting the bottlenecks that hinder the exploitation of a large number of CPU cores has not been an easy task. A tool that would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

http://release.softlab.ntua.gr/bencherl/



Server with four Intel Xeon E7340 CPUs (2.40 GHz) and 8GB of RAM running Linux 2.6.32-5-amd64 (a total of 16 cores).
Image 3 of 6

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.

Scalability of ETS: R11 to R16 …

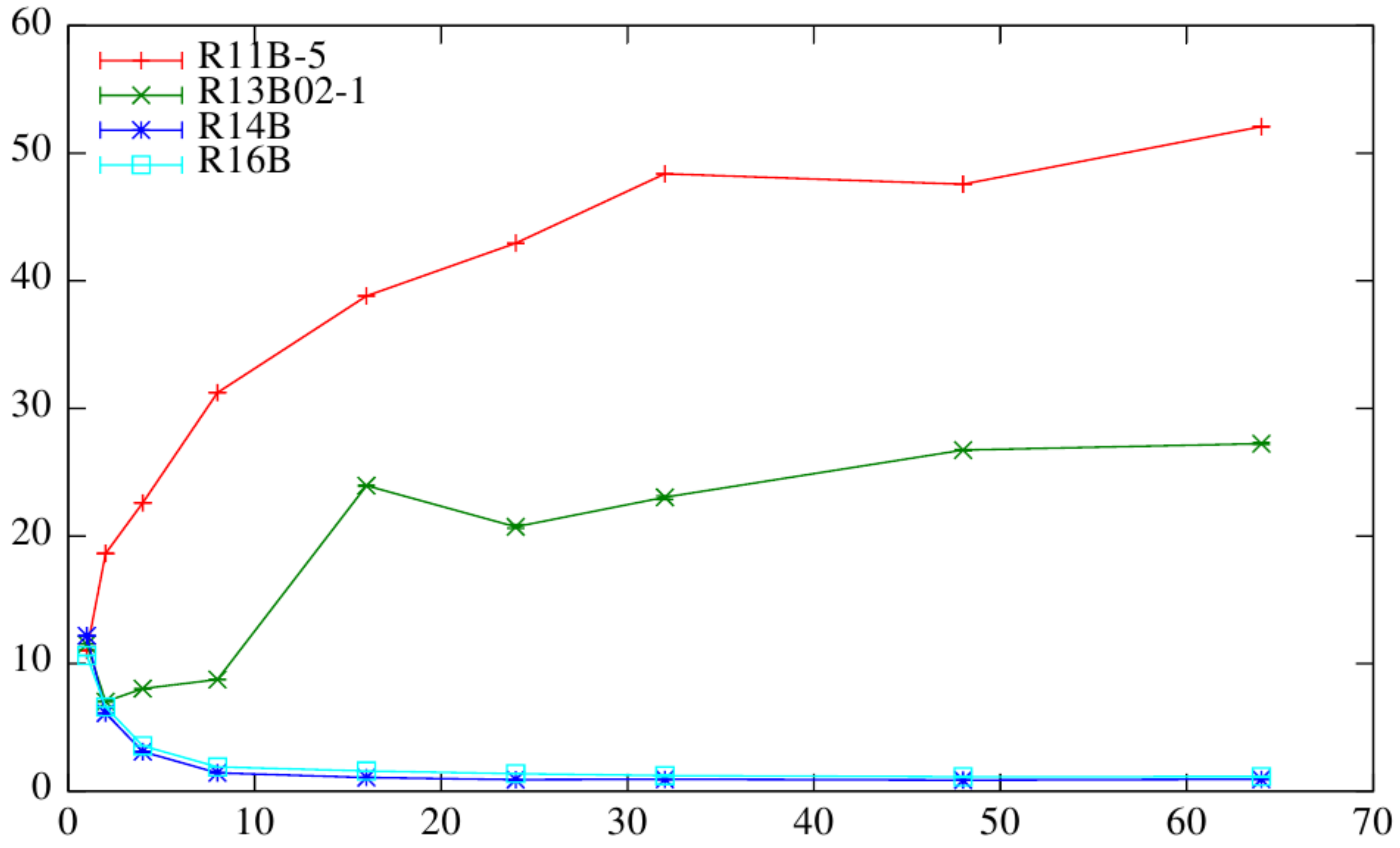


Figure 6. Scalability of ETS tables of type `set` across Erlang/OTP releases using a workload with 99% lookups and 1% updates.

Scalability of ETS: R11 to R16 …

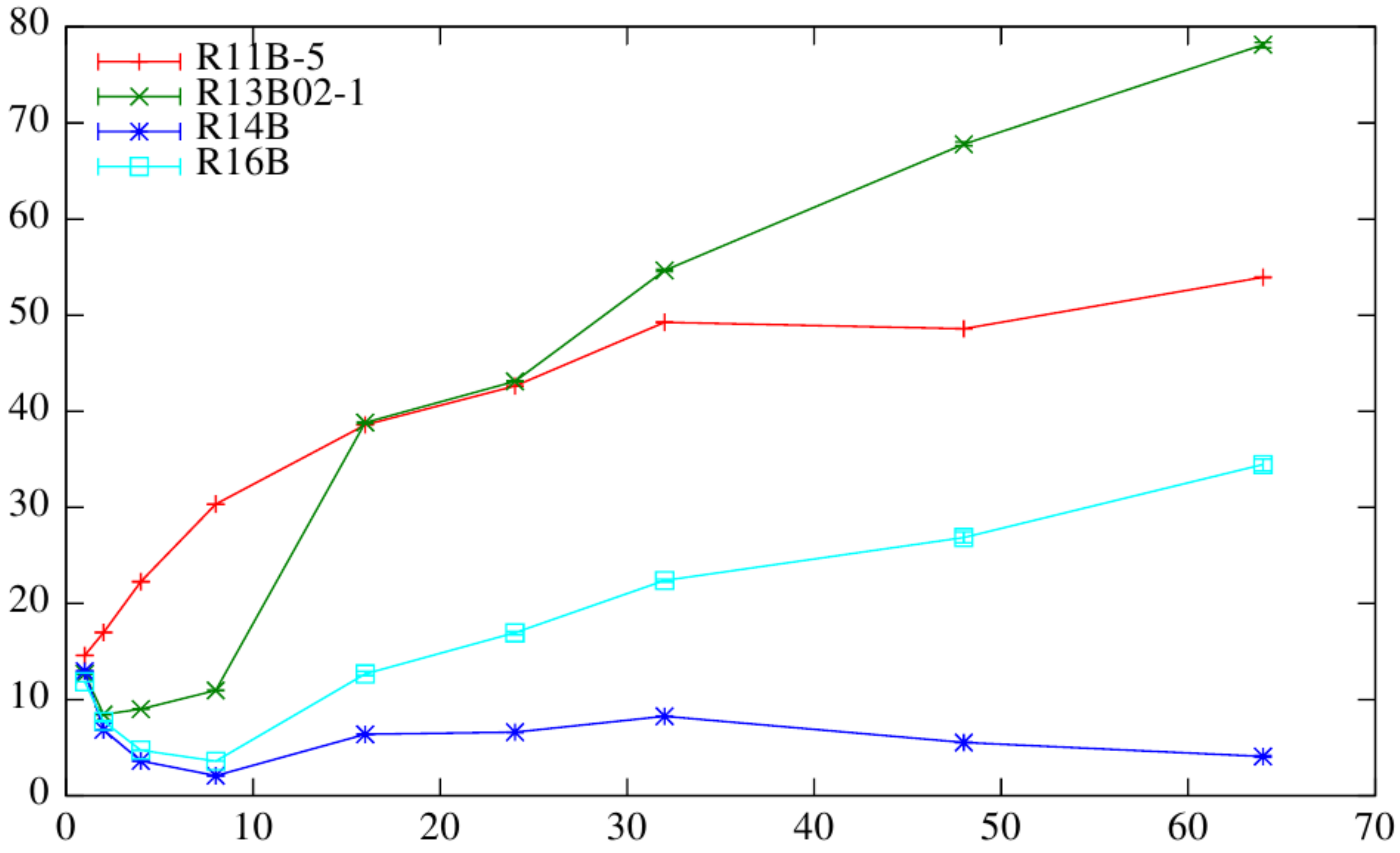


Figure 7. Scalability of ETS tables of type `ordered_set` across OTP releases using a workload with 99% lookups and 1% updates.

Concurrency options R16 …

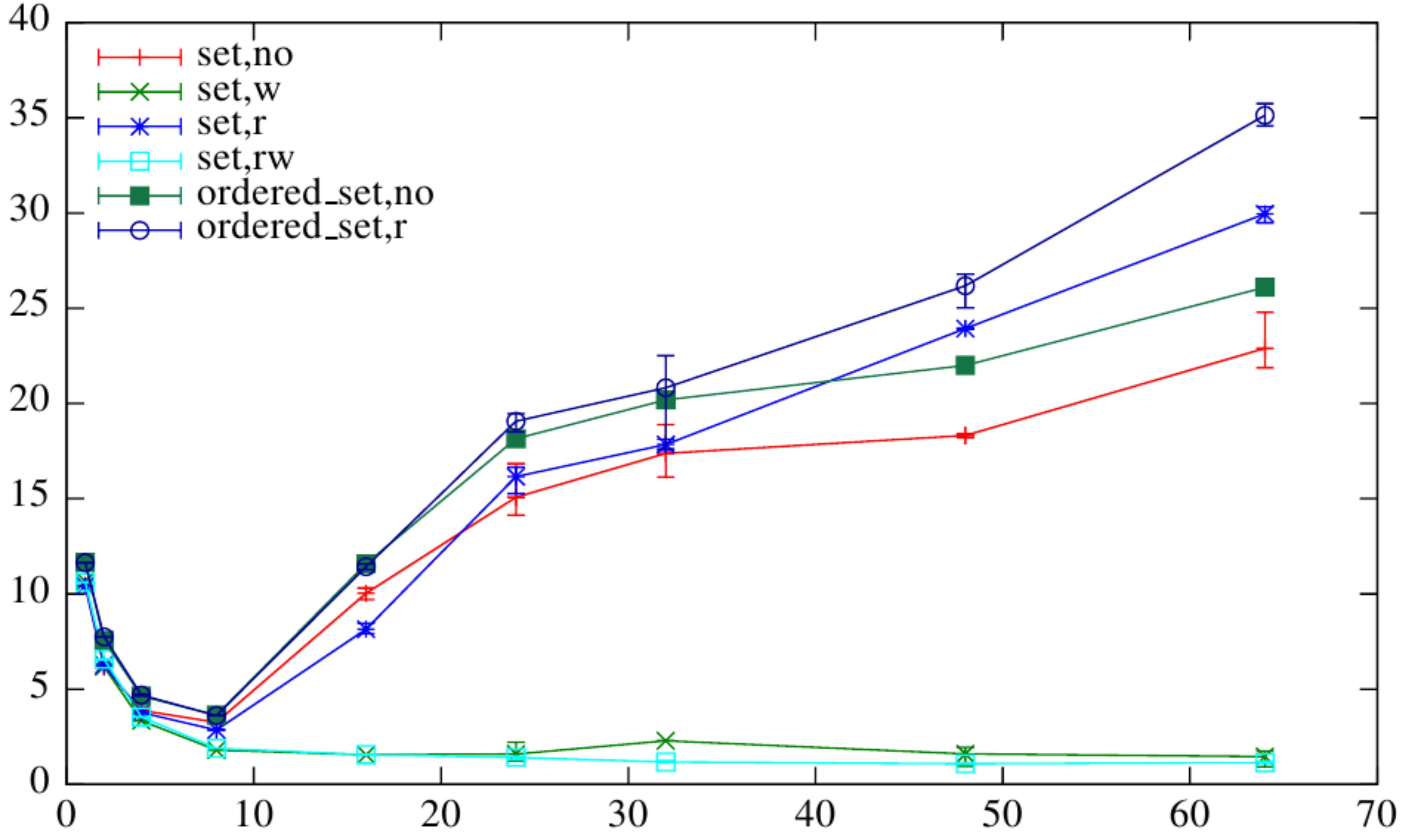


Figure 11. Scalability of ETS on a workload with 99% lookups and 1% updates when varying the ETS table concurrency options.

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

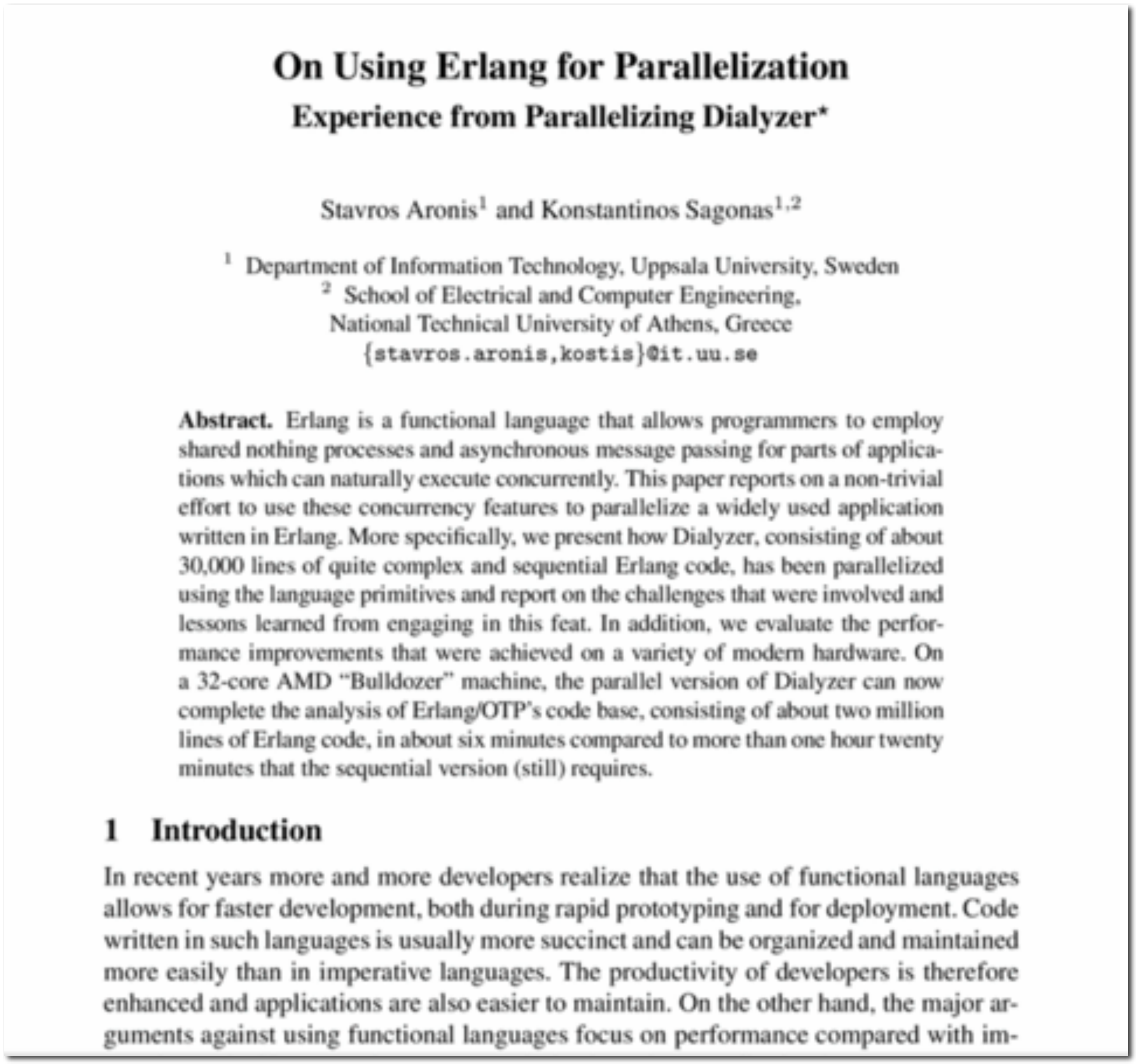
- Use pinning on NUMA
- Use `read_concurrency` when doing only lookups
- Use `write_concurrency`
- Measure your use case when combining them

Eating our own dog food …

Applied the techniques of the project to our own systems …

… Dialyzer, and …

… Wrangler.



On Using Erlang for Parallelization

Experience from Parallelizing Dialyzer*

Stavros Aronis[1] and Konstantinos Sagonas[1,2]

[1] Department of Information Technology, Uppsala University, Sweden
[2] School of Electrical and Computer Engineering, National Technical University of Athens, Greece
{stavros.aronis,kostis}@it.uu.se

Abstract. Erlang is a functional language that allows programmers to employ shared nothing processes and asynchronous message passing for parts of applications which can naturally execute concurrently. This paper reports on a non-trivial effort to use these concurrency features to parallelize a widely used application written in Erlang. More specifically, we present how Dialyzer, consisting of about 30,000 lines of quite complex and sequential Erlang code, has been parallelized using the language primitives and report on the challenges that were involved and lessons learned from engaging in this feat. In addition, we evaluate the performance improvements that were achieved on a variety of modern hardware. On a 32-core AMD "Bulldozer" machine, the parallel version of Dialyzer can now complete the analysis of Erlang/OTP's code base, consisting of about two million lines of Erlang code, in about six minutes compared to more than one hour twenty minutes that the sequential version (still) requires.

1 Introduction

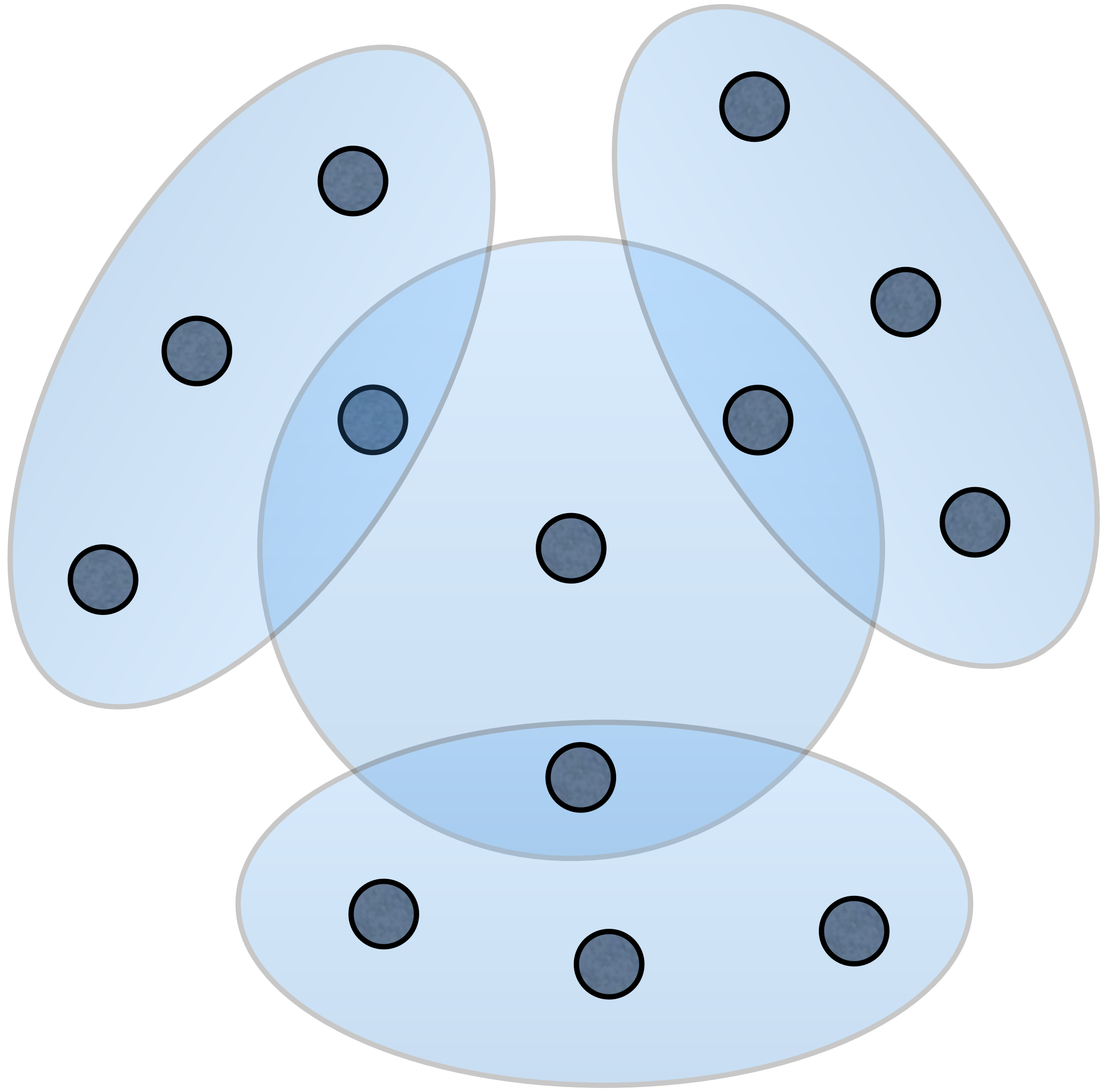
In recent years more and more developers realize that the use of functional languages allows for faster development, both during rapid prototyping and for deployment. Code written in such languages is usually more succinct and can be organized and maintained more easily than in imperative languages. The productivity of developers is therefore enhanced and applications are also easier to maintain. On the other hand, the major arguments against using functional languages focus on performance compared with im-

SD Erlang

Scalable distribution: SD Erlang

Patterns for interconnection.

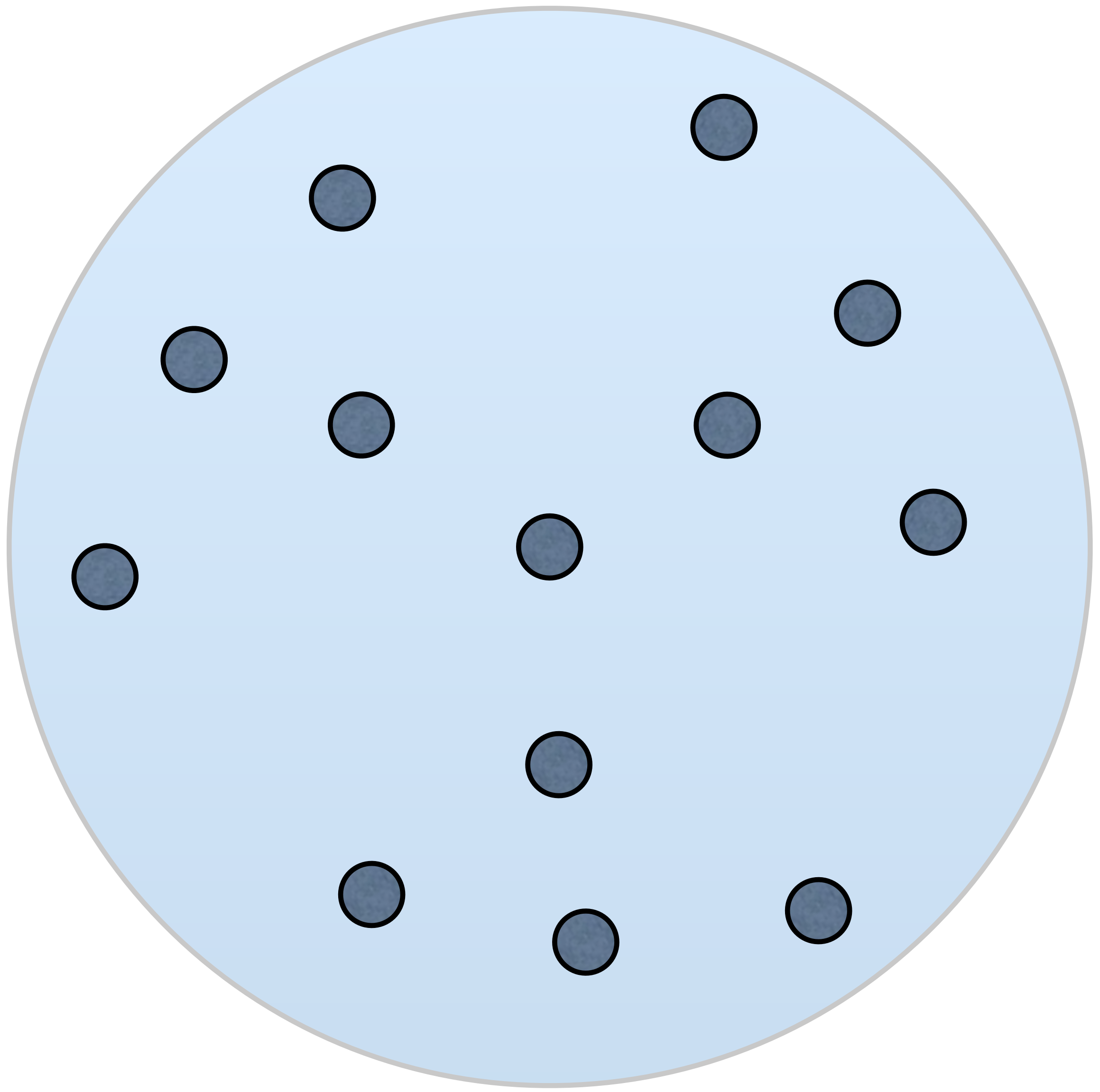
Semi-explicit process deployment.



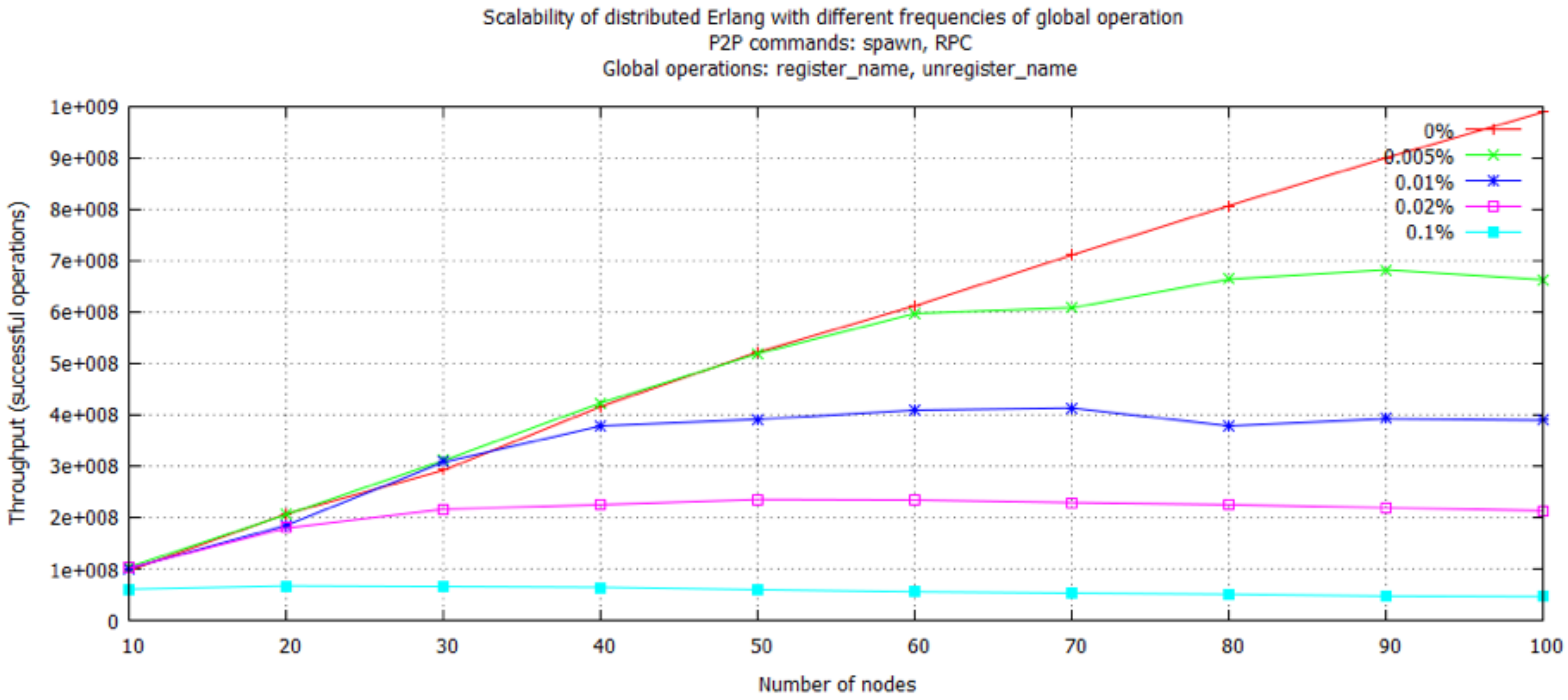
Distribution "out of the box"

Completely connected: all nodes connected to each other.

Quadratic complexity.

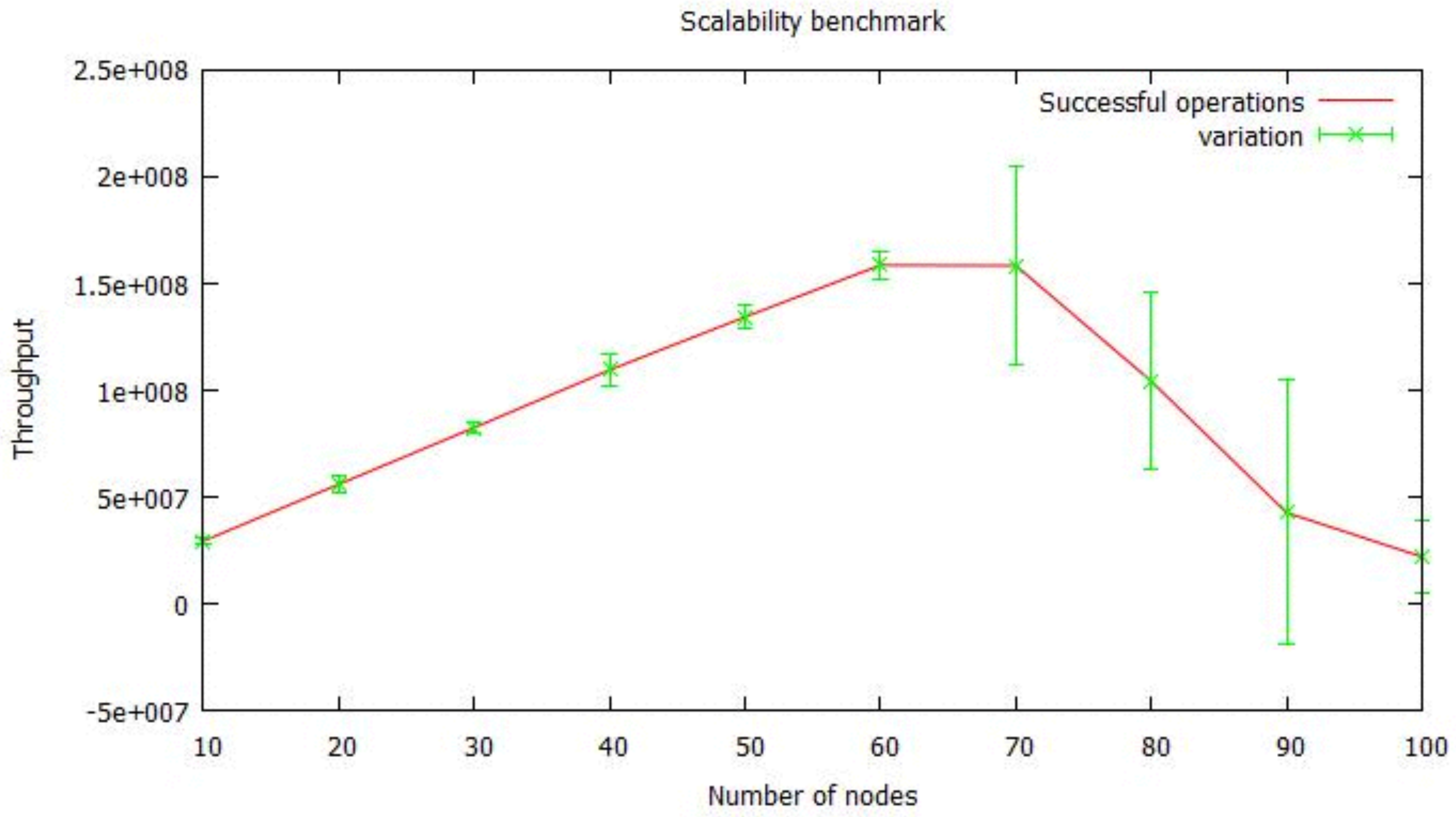


Scalability



Scalability

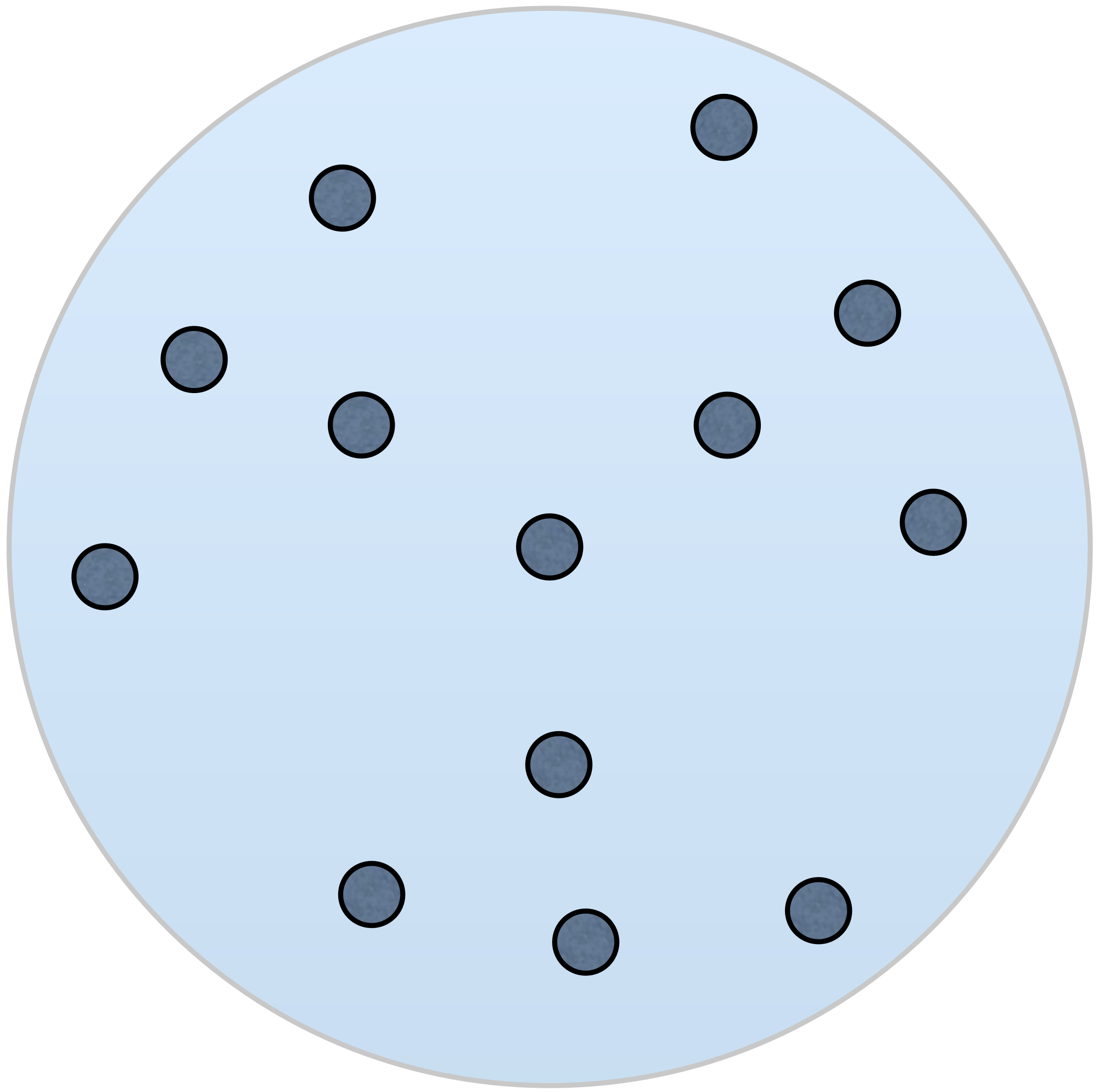
http://www.dcs.gla.ac.uk/~amirg/publications/ScalablePersistentStorage.pdf



Distribution “out of the box”

Completely connected: all nodes connected to each other.

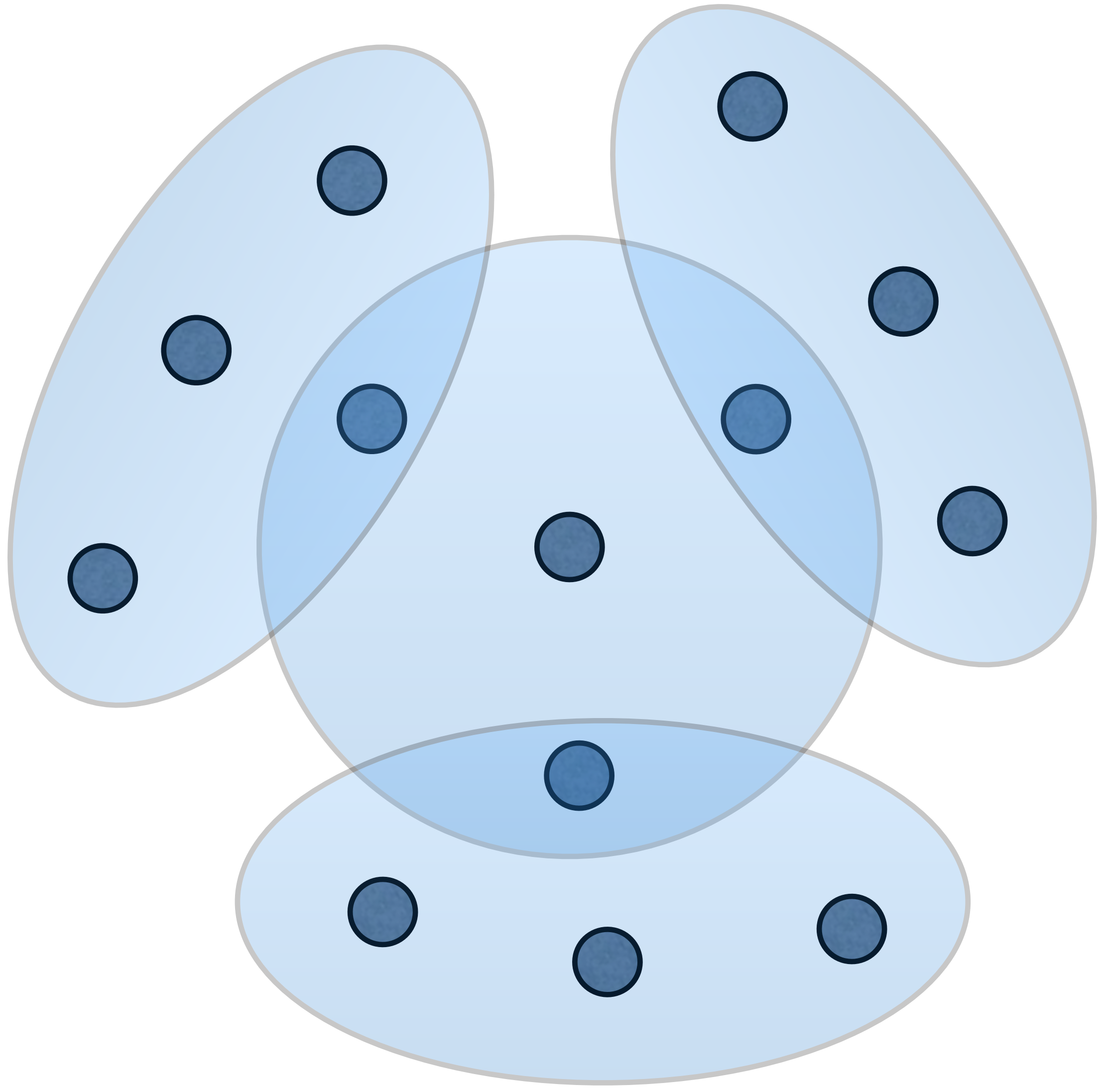
Quadratic complexity.



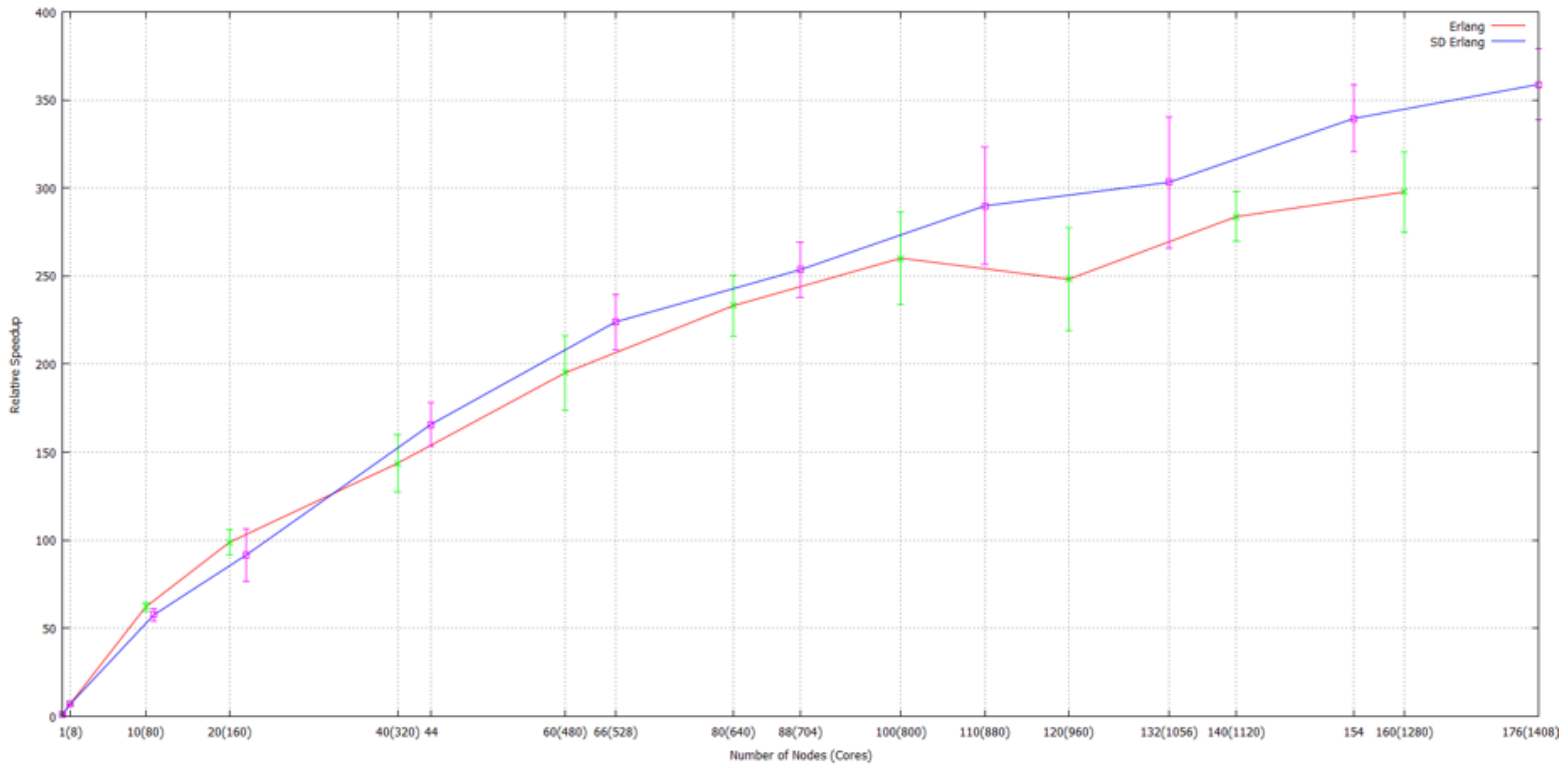
SD Erlang "out of the box"

Complete connectivity within each s_group.

Overlap topology supports nesting, hierarchy and *ad hoc* models.

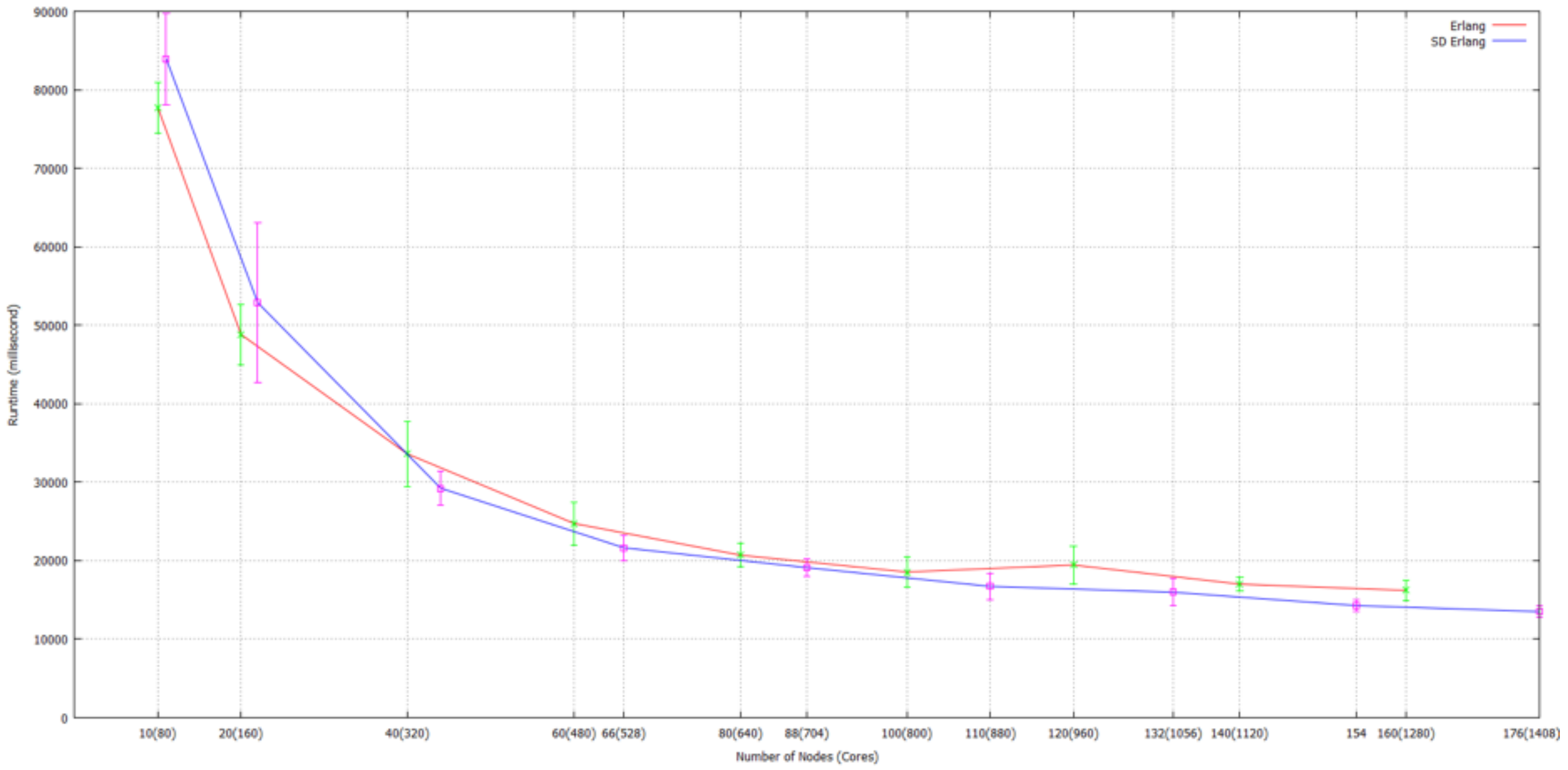


Speedup

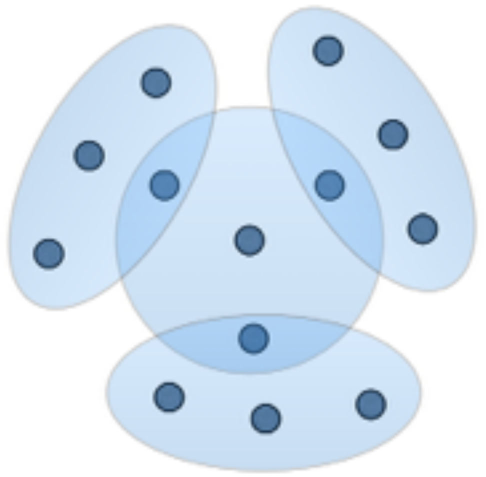


Erlang
SD Erlang
Relative Speedup
Number of Nodes (Cores)
0
50
100
150
200
250
300
350
400
1(8)
10(80)
20(160)
40(320)
44
60(480)
66(528)
80(640)
88(704)
100(800)
110(880)
120(960)
132(1056)
140(1120)
154
160(1280)
176(1408)

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.

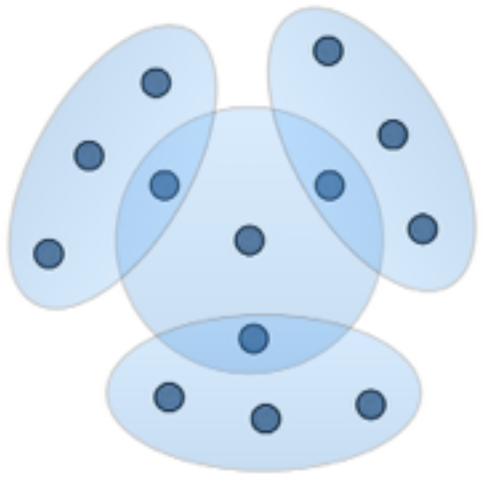
Register, re-register and unregister names in an s_group.

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}])
```

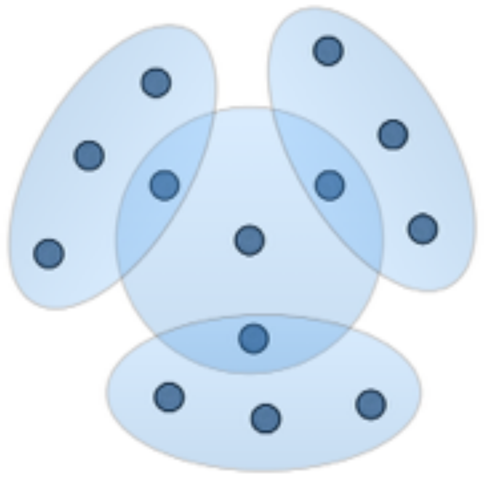
Choose eligible nodes for spawn from the identified s_group.

```
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

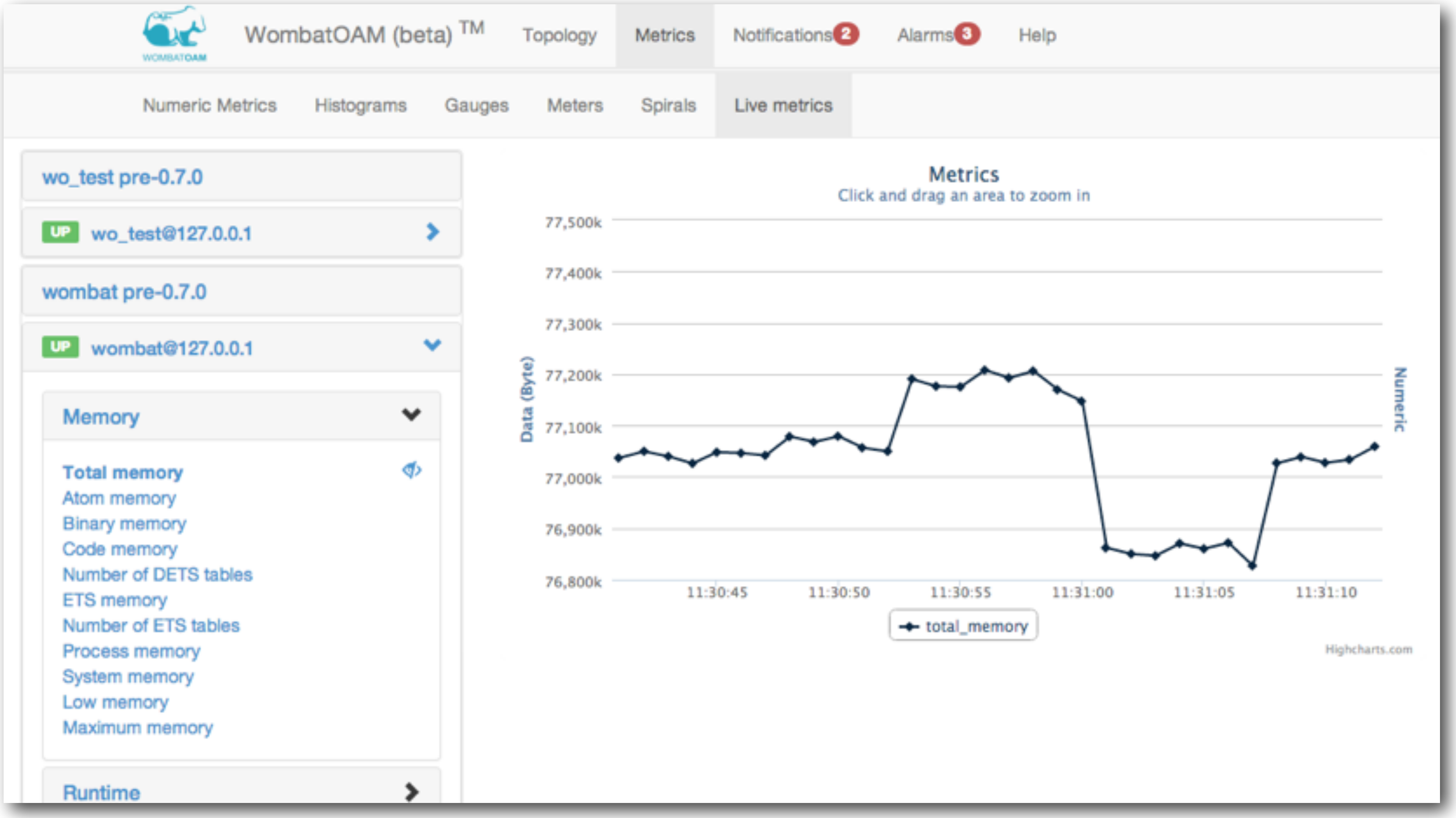
WombatOAM

WombatOAM is an operations and maintenance framework for Erlang based systems.

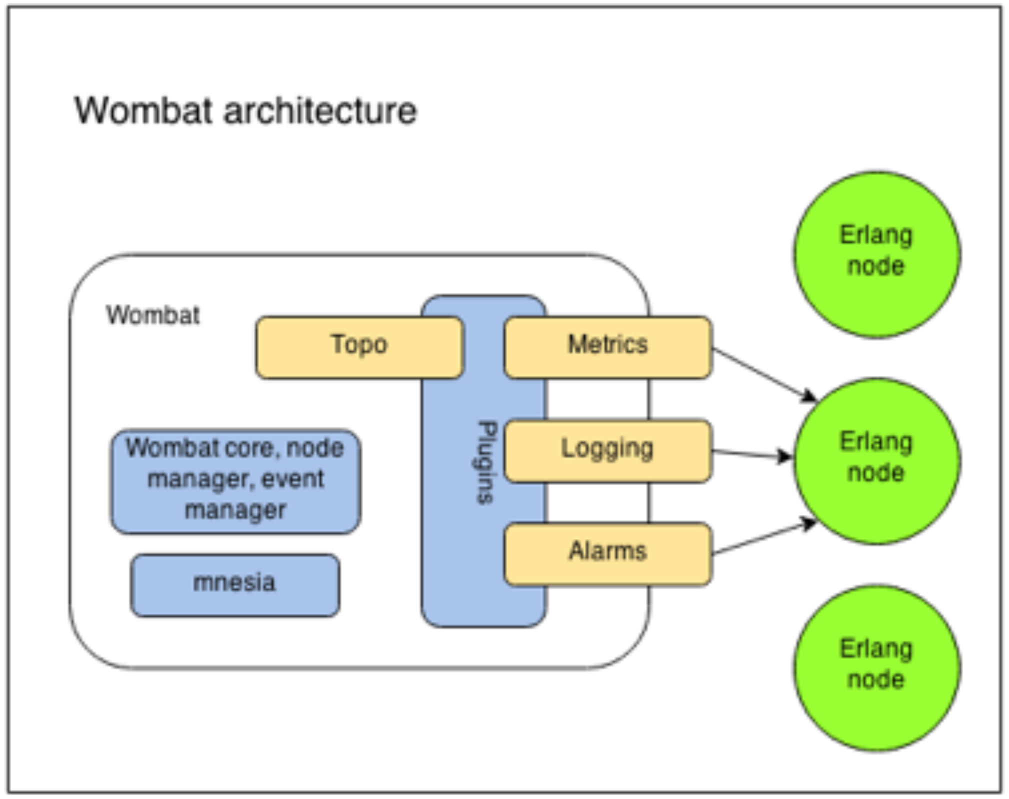
It gives you full visibility on what is going on in Erlang clusters …

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

How it looks


WombatOAM (beta) TM
Topology
Metrics
Notifications 2
Alarms 3
Help
Numeric Metrics
Histograms
Gauges
Meters
Spirals
Live metrics
wo_test pre-0.7.0
UP wo_test@127.0.0.1
wombat pre-0.7.0
UP wombat@127.0.0.1
Memory
Total memory
Atom memory
Binary memory
Code memory
Number of DETS tables
ETS memory
Number of ETS tables
Process memory
System memory
Low memory
Maximum memory
Runtime
Metrics
Click and drag an area to zoom in
77,500k
77,400k
77,300k
77,200k
77,100k
77,000k
76,900k
76,800k
Data (Byte)
Numeric
11:30:45
11:30:50
11:30:55
11:31:00
11:31:05
11:31:10
total_memory
Highcharts.com

WombatOAM



Monitor managed nodes liveliness

Group nodes by Erlang releases

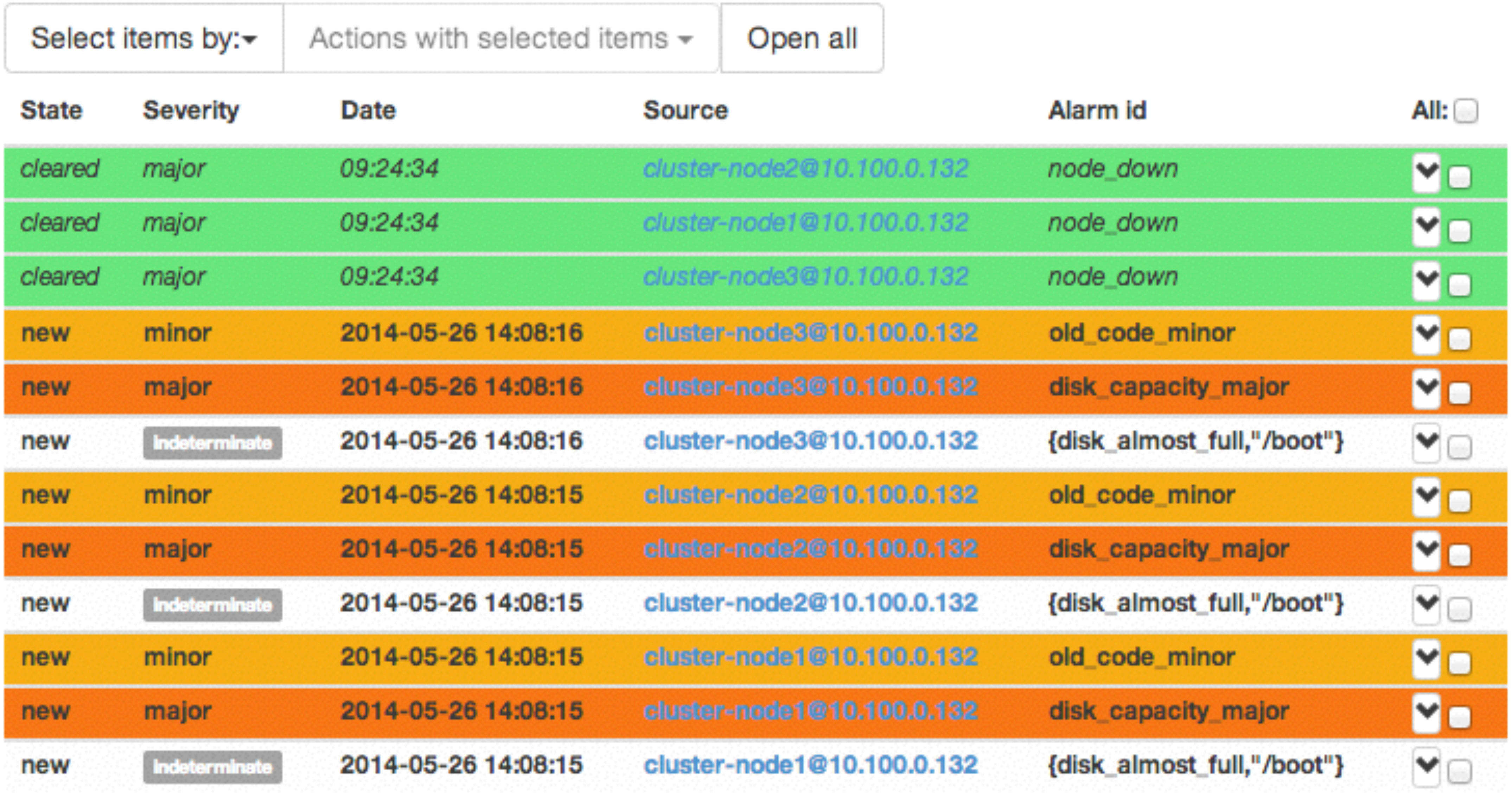
Deploy Erlang releases in the cloud

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



State	Severity	Date	Source	Alarm id	All:
cleared	*major*	*09:24:34*	*cluster-node2@10.100.0.132*	*node_down*	
cleared	*major*	*09:24:34*	*cluster-node1@10.100.0.132*	*node_down*	
cleared	*major*	*09:24:34*	*cluster-node3@10.100.0.132*	*node_down*	
new	minor	2014-05-26 14:08:16	cluster-node3@10.100.0.132	old_code_minor	
new	major	2014-05-26 14:08:16	cluster-node3@10.100.0.132	disk_capacity_major	
new	indeterminate	2014-05-26 14:08:16	cluster-node3@10.100.0.132	{disk_almost_full,"/boot"}	
new	minor	2014-05-26 14:08:15	cluster-node2@10.100.0.132	old_code_minor	
new	major	2014-05-26 14:08:15	cluster-node2@10.100.0.132	disk_capacity_major	
new	indeterminate	2014-05-26 14:08:15	cluster-node2@10.100.0.132	{disk_almost_full,"/boot"}	
new	minor	2014-05-26 14:08:15	cluster-node1@10.100.0.132	old_code_minor	
new	major	2014-05-26 14:08:15	cluster-node1@10.100.0.132	disk_capacity_major	
new	indeterminate	2014-05-26 14:08:15	cluster-node1@10.100.0.132	{disk_almost_full,"/boot"}	

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.

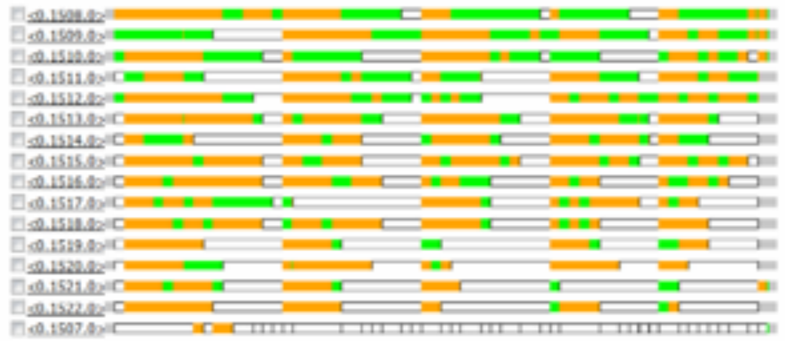
Concuerror

Systematic concurrency testing of Erlang programs.

Case studies for Mochiweb and Poolboy.

http://concuerror.com

Percept2



Profile … analyse … display in a browser, enhancing Percept.

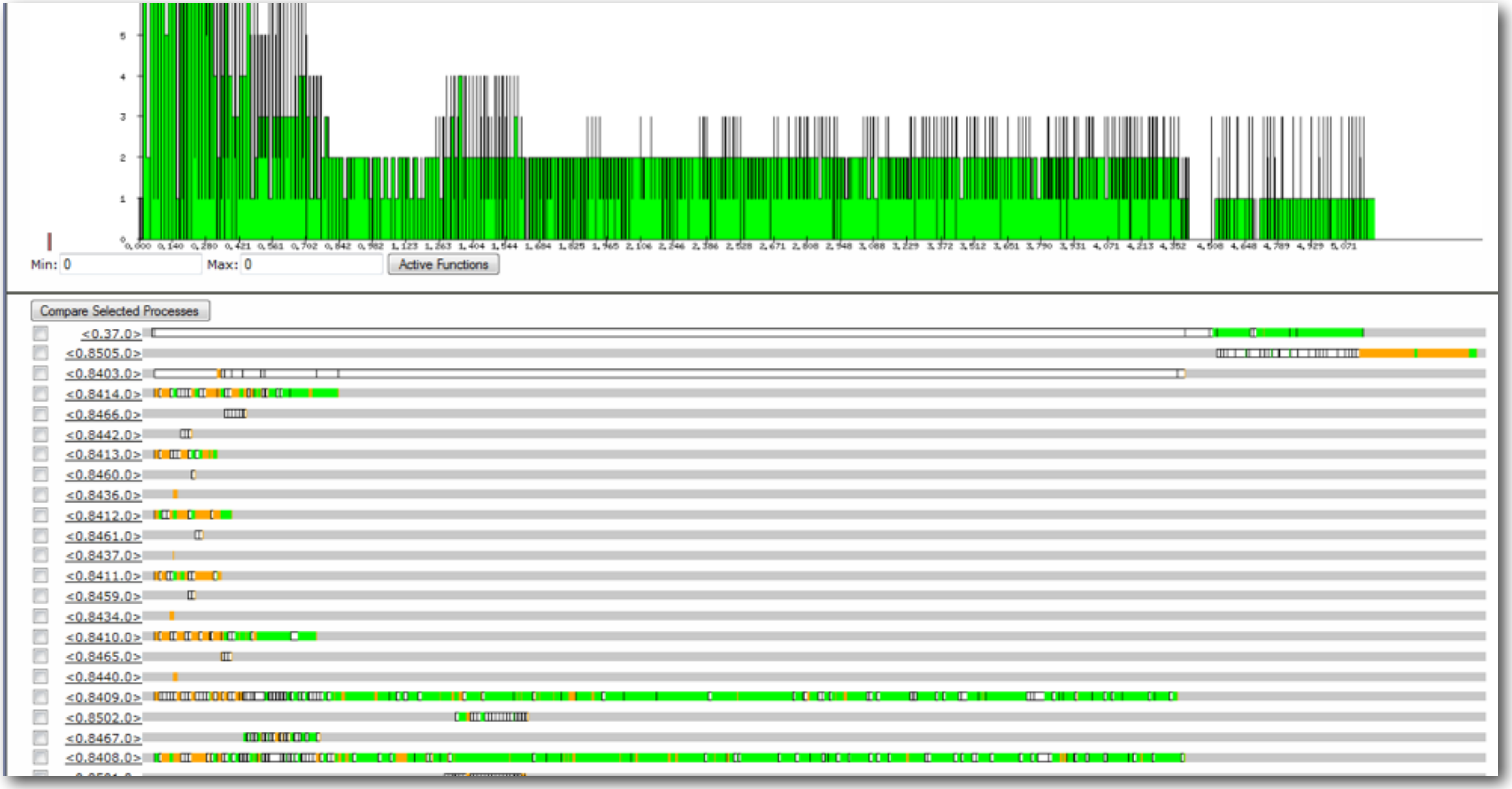
Percept: active processes vs. time, drill down to process info …
… including runnability, start/end time, parent/child processes, etc.

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

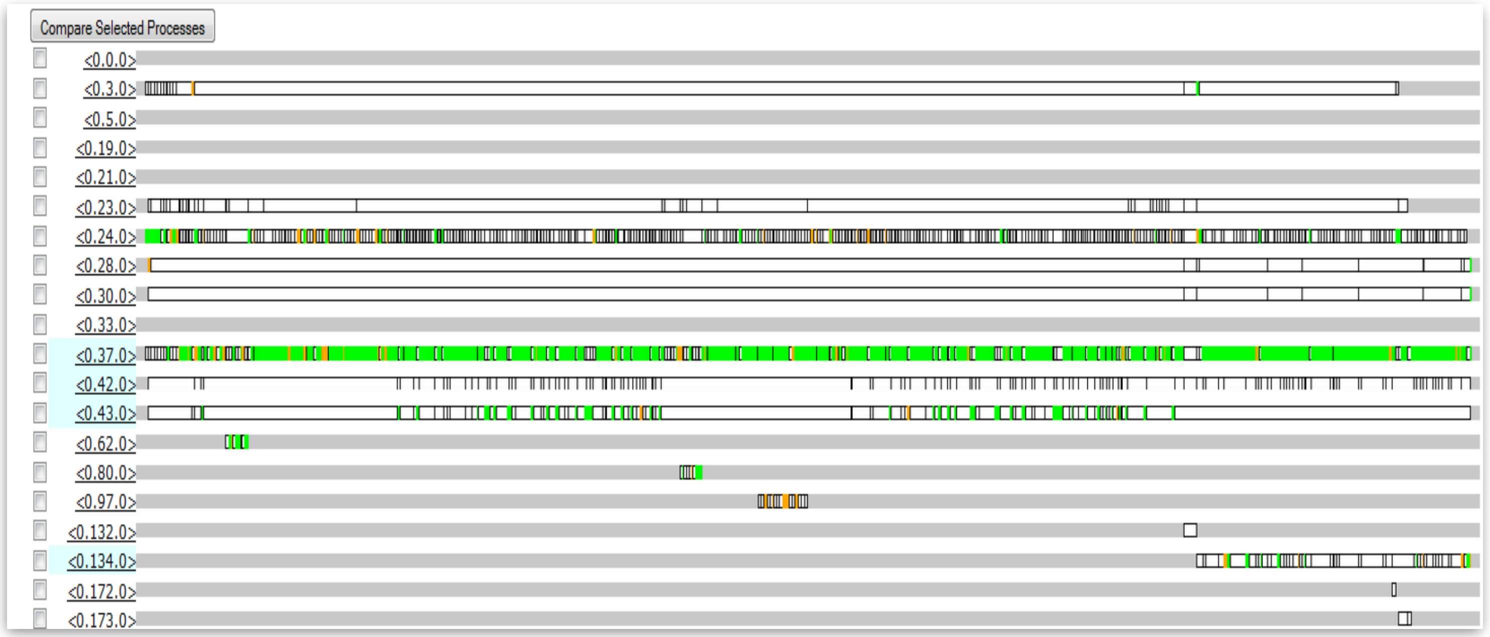
Scalability: scalable process tree, selective profiling, parallel analysis and caching history webpages.

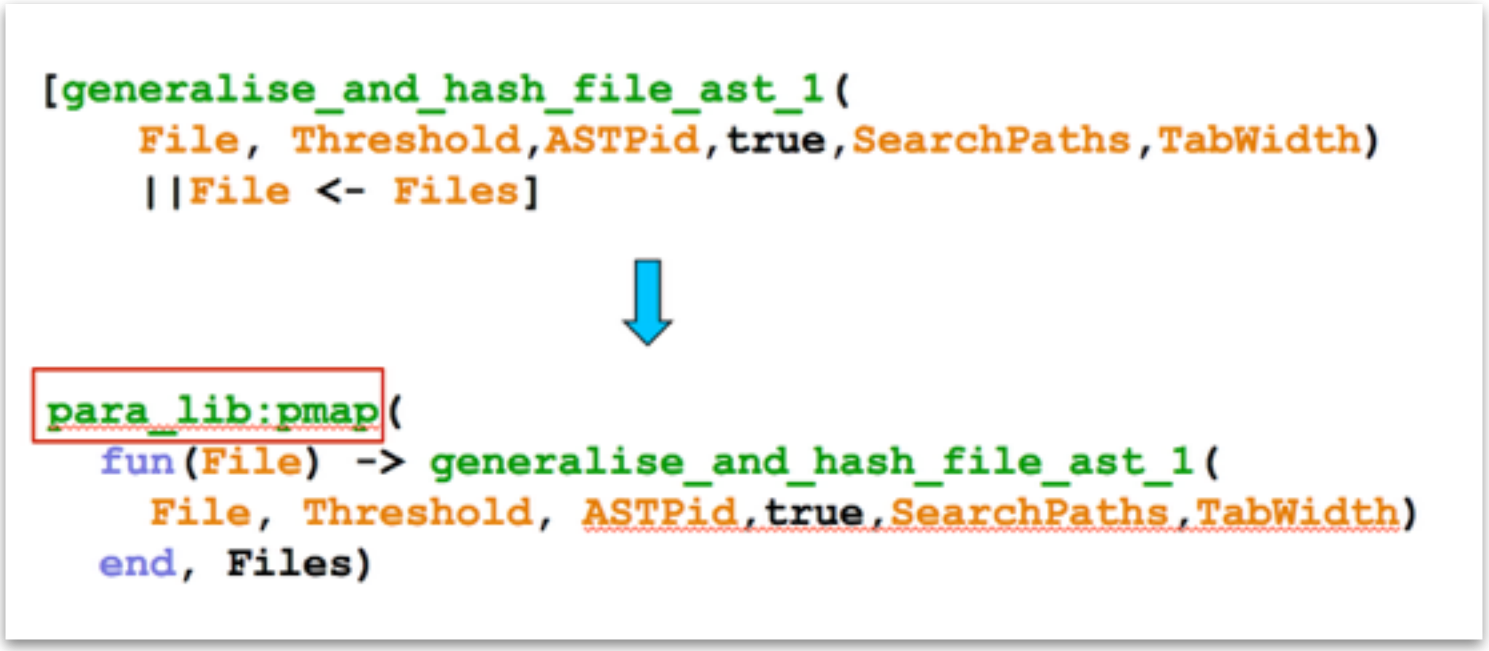
https://github.com/RefactoringTools/percept2

Improving Wrangler using Percept2



Improving Wrangler using Percept2

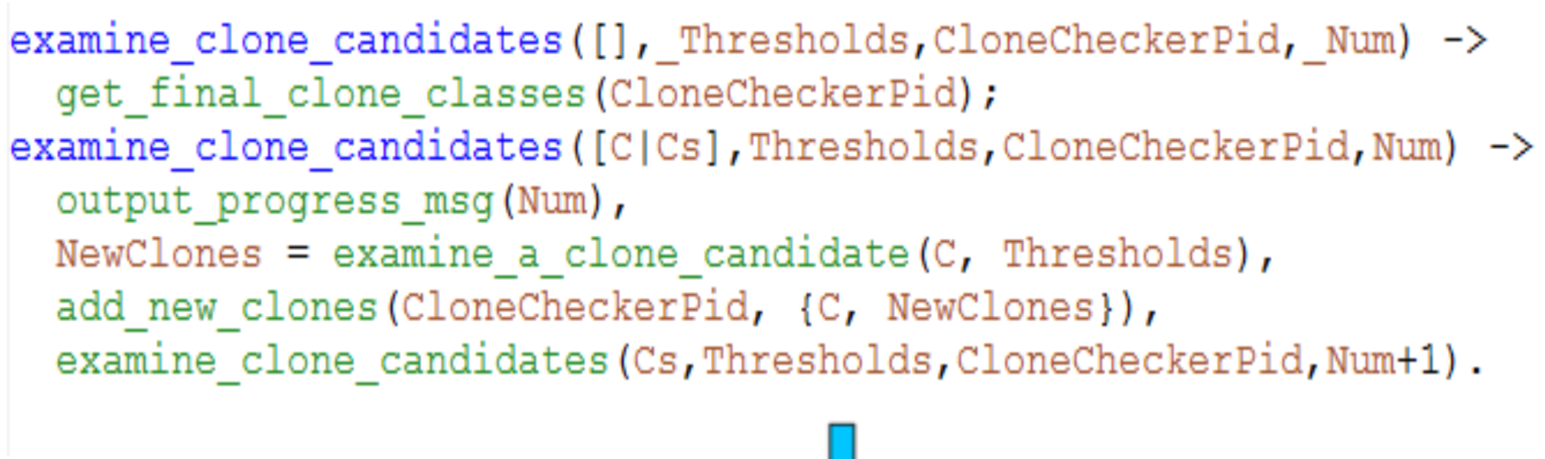




```
[generalise_and_hash_file_ast_1(
    File, Threshold,ASTPid,true,SearchPaths,TabWidth)
    ||File <- Files]
```

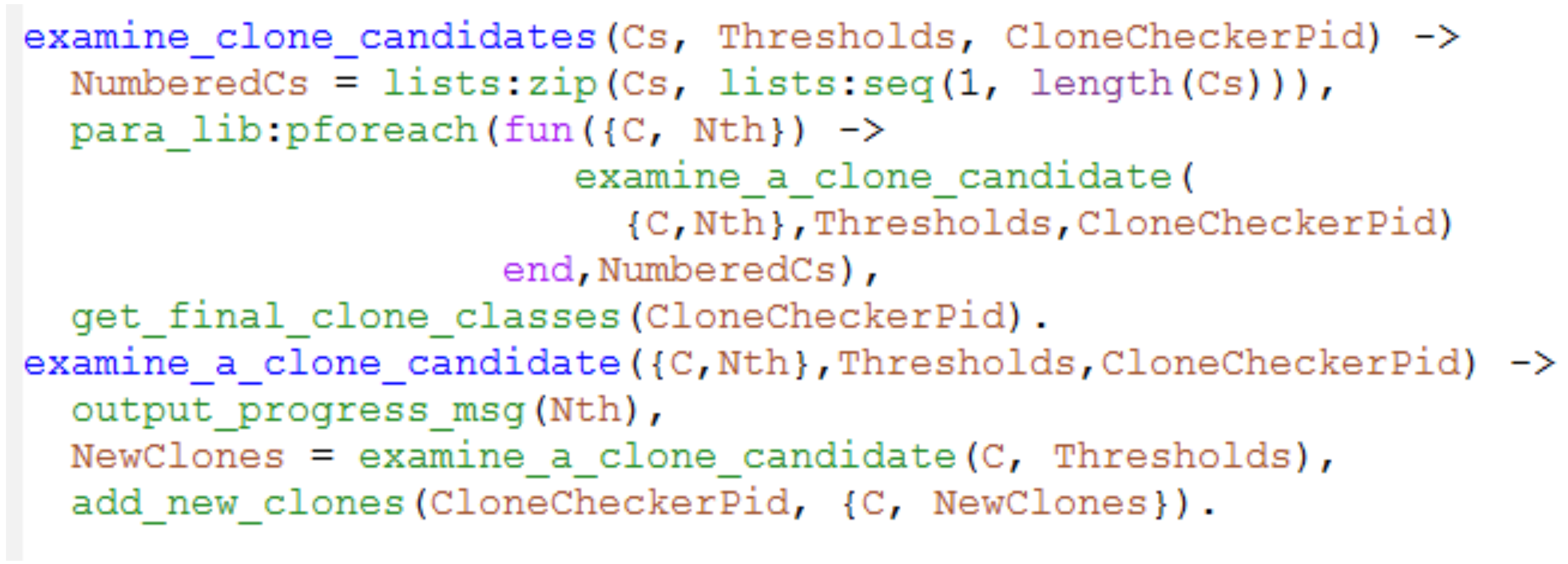
```
para_lib:pmap(
  fun(File) -> generalise_and_hash_file_ast_1(
    File, Threshold, ASTPid,true,SearchPaths,TabWidth)
  end, Files)
```

Improving Wrangler using Percept2



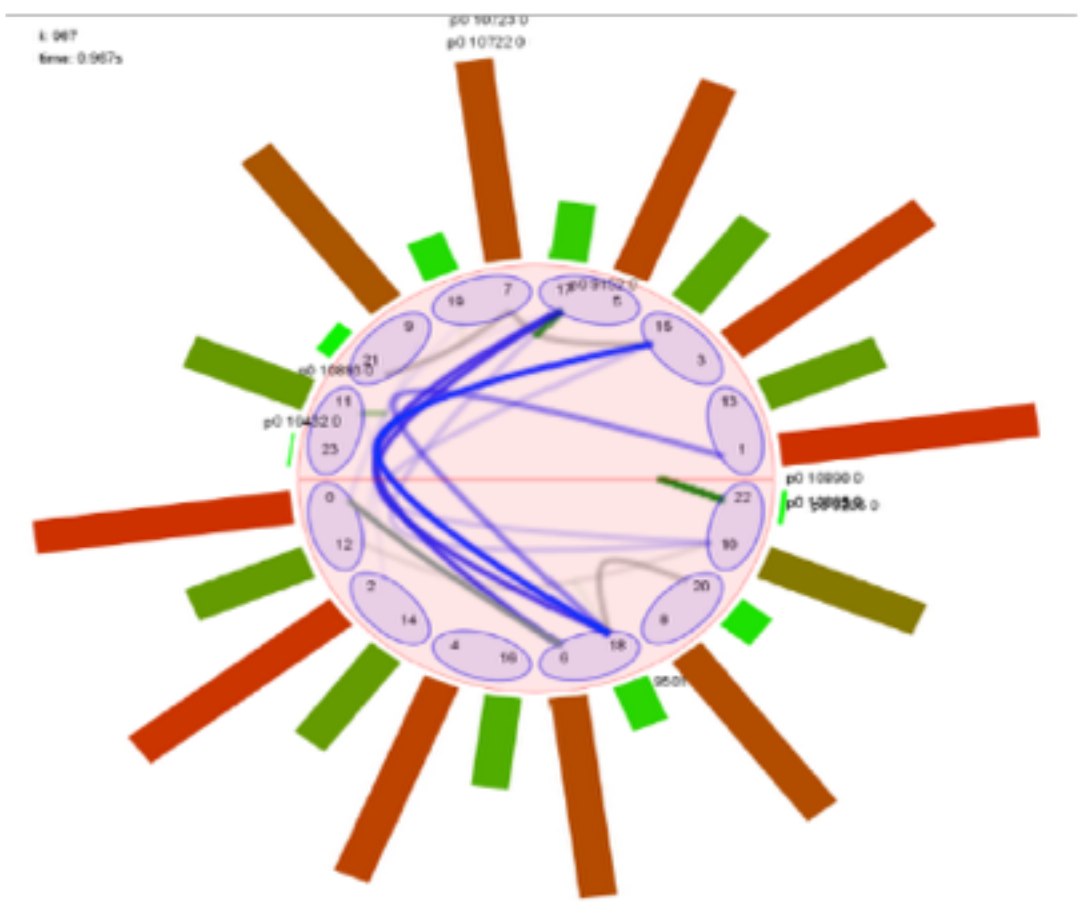
```erlang
examine_clone_candidates([],_Thresholds,CloneCheckerPid,_Num) ->
  get_final_clone_classes(CloneCheckerPid);
examine_clone_candidates([C|Cs],Thresholds,CloneCheckerPid,Num) ->
  output_progress_msg(Num),
  NewClones = examine_a_clone_candidate(C, Thresholds),
  add_new_clones(CloneCheckerPid,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

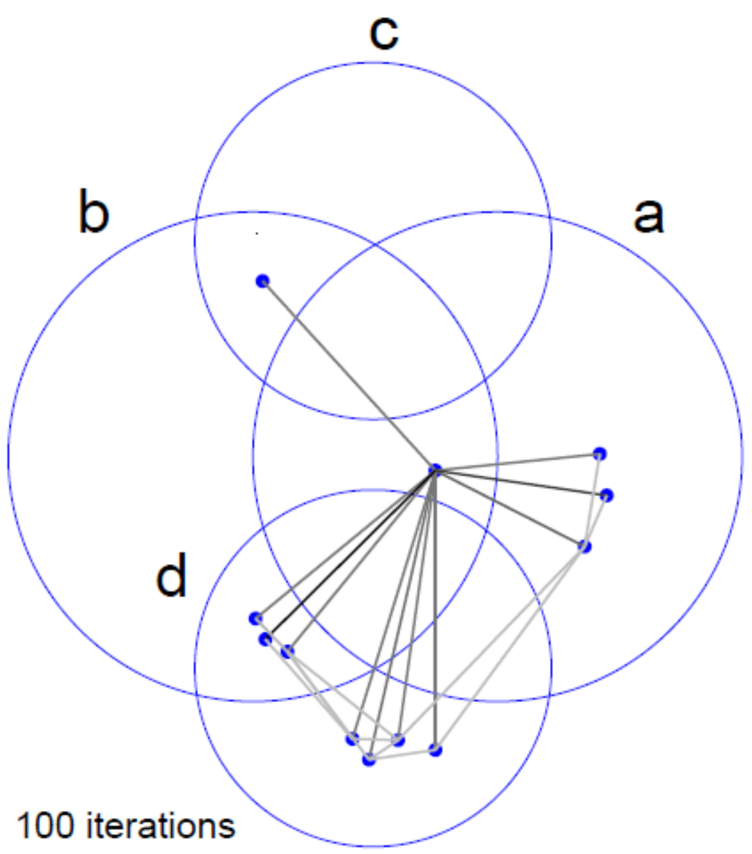
⇩



```erlang
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

Sim-Diasca

Simulation of Discrete Systems of All Scales

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

RELEASE

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?