The Design of Scalable Distributed (SD) Erlang

Natalia Chechina, Phil Trinder,

Amir Ghaffari, Rickard Green, Kenneth Lundin, and Robert Verding

September 1, 2012

Outline

- Background
- Motivation & Challenges
- Scalable Distributed (SD) Erlang Design
- Conclusion and Future work

RELEASE

RELEASE Project

- Aim - Scaling the **radical concurrency-oriented programming paradigm** to build **reliable general-purpose software** on **massively parallel machines**
- Working at three levels
  - Evolving the Erlang VM
  - Evolving the language to Scalable Distributed (SD) Erlang
  - Developing a scalable Erlang infrastructure

RELEASE

Erlang

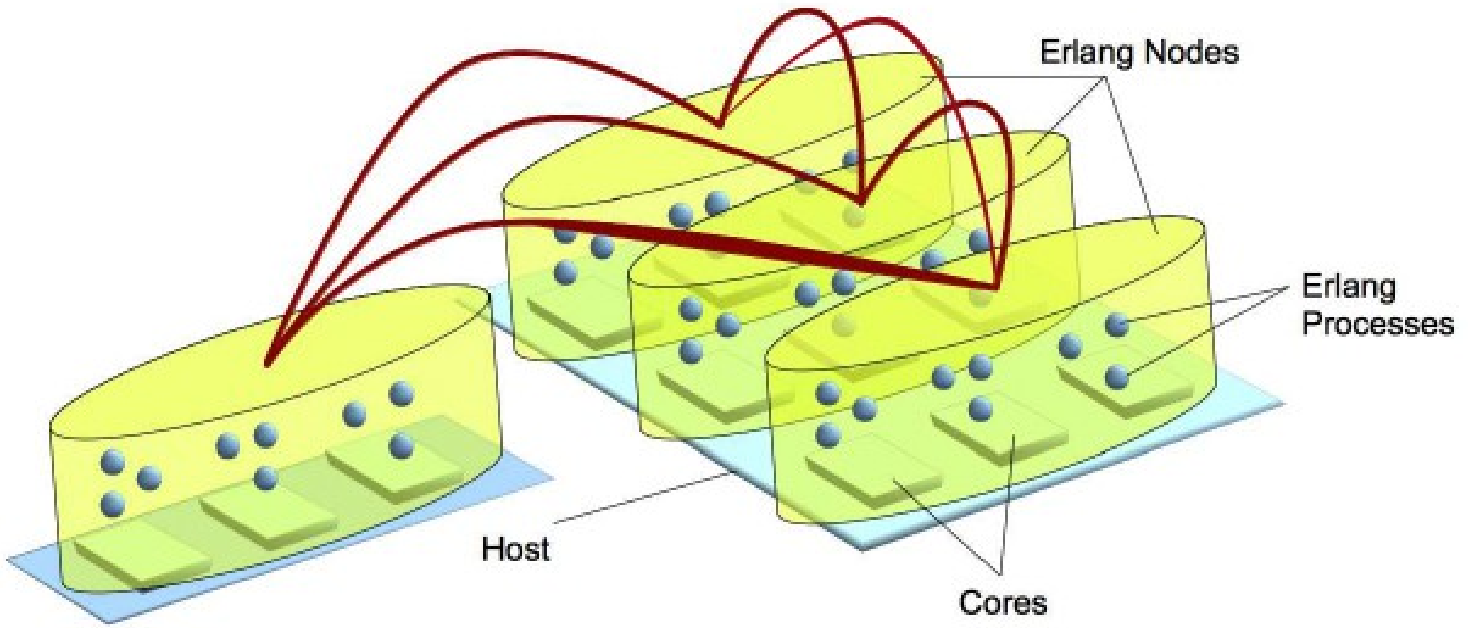
- Erlang is a functional actor-based concurrent dynamically typed general purpose programming language
- Erlang was designed in 1986 for
    - Distributed
    - Fault-tolerant
    - Massively concurrent
    - Soft-real time systems
- Concurrency is handled by the language and not by the operating system

Erlang Philosophy

- Share nothing
  - Processes are isolate
  - Processes do not share memory
  - Variables are not reusable
- Let it Crash
  - Non-defensive approach
  - Processes crash
  - Other processes detect and fix the problem

RELEASE

Distributed Erlang & Motivation

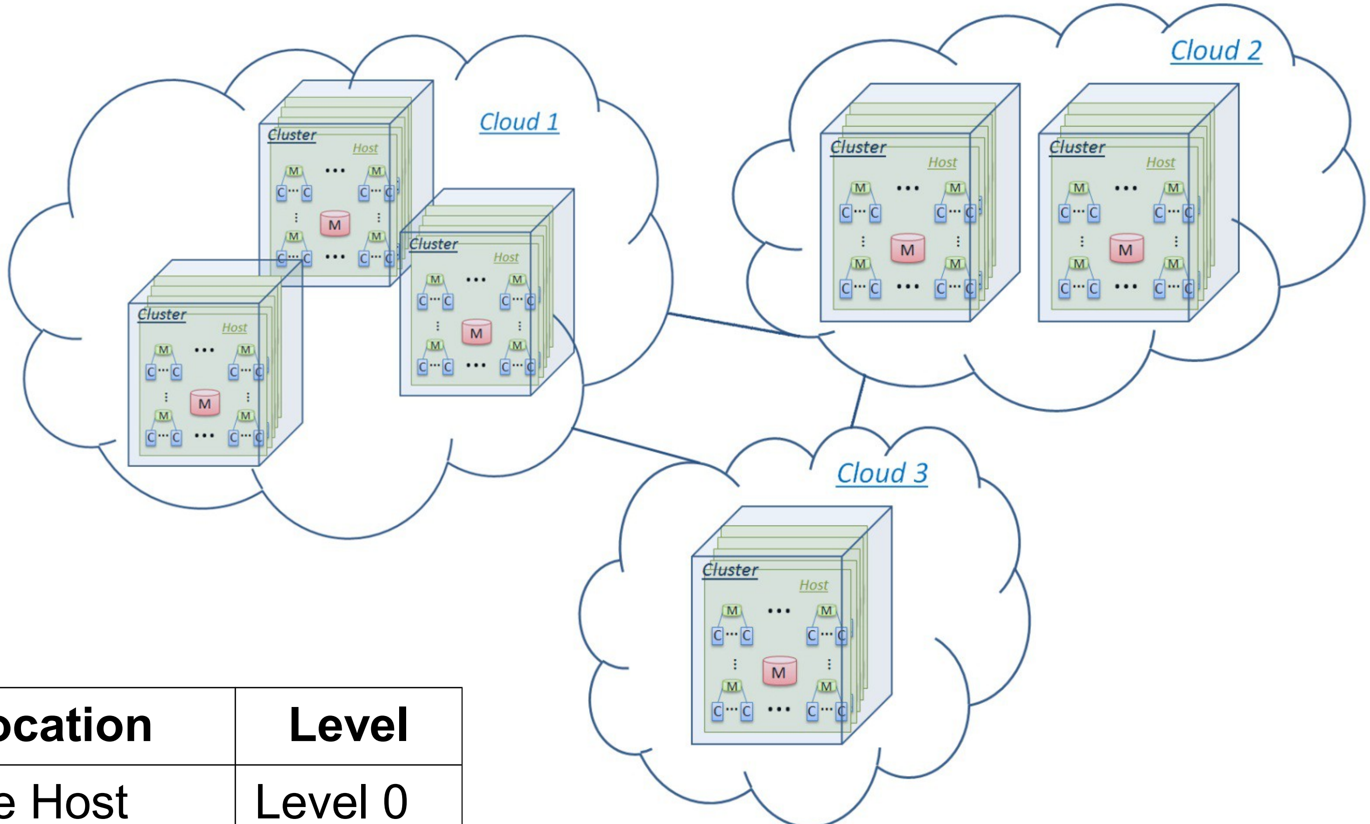


1) Transitive connections

2) Explicit placement

RELEASE

Typical architecture – 10^5 cores



Location	Level
Same Host	Level 0
Same Cluster	Level 1
Same Cloud	Level 2
Another Cloud	Level 3

- Commodity hardware
- Non-uniform communication

RELEASE

Scaling

- Persistent data structures
    - Riak, Casandra – P2P key/value database systems
- In-memory data structures
    - ETS tables
- Computation

RELEASE

Challenges

- Provide scalability while preserving Erlang's reliability mechanisms & supervision behaviours
- SD Erlang to become a part of Erlang distribution

RELEASE

General Design Principles

- Working at Erlang level as far as possible
- Preserving the Erlang philosophy and programming idioms
- Minimal language changes

RELEASE

Reliable Scalability Design Principles

- Avoiding global sharing
- Avoiding explicit prescription
- Introducing an abstract notion of communication architecture
- Keeping Erlang reliability model unchanged as far as possible

RELEASE

SD Erlang Design Directions

- Network Scalability
  - All to all connections are not scalable onto 1000s of nodes
  - Aim: Reduce connectivity
- Semi-explicit Placement
  - Becomes not feasible for a programmer to be aware of all nodes
  - Aim: Automatic process placement in groups of nodes

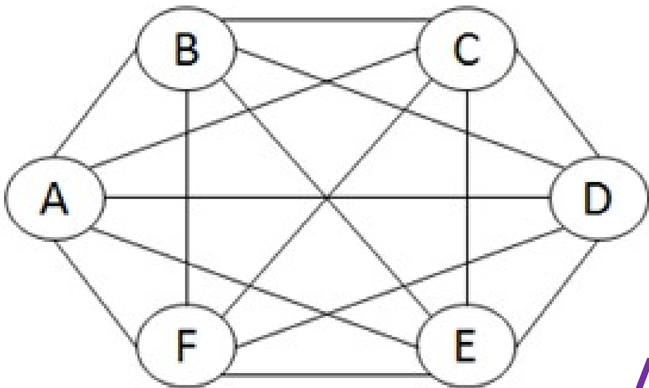
RELEASE

Network Scalability

- Grouping nodes in Scalable groups (s_groups)
  - **transitive** connections with nodes of the same s_group
  - **non-transitive** connections with other nodes
- Types of s_groups:
  - Hierarchical
  - **Overlapping**
  - Partition
- Using **s_group** names instead of **global** names: *Name@Group*

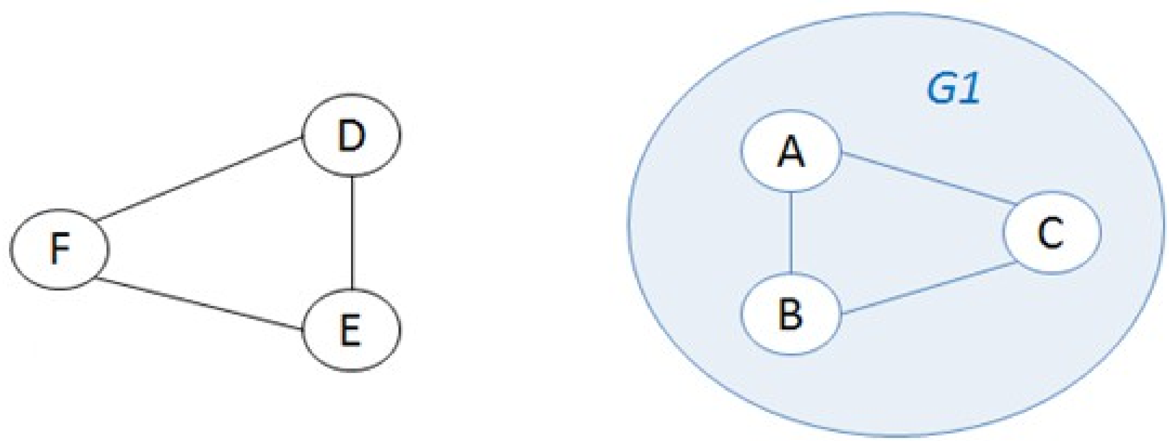
RELEASE

Creating an s_group



A: new_s_group(G1, [A, B, C]).

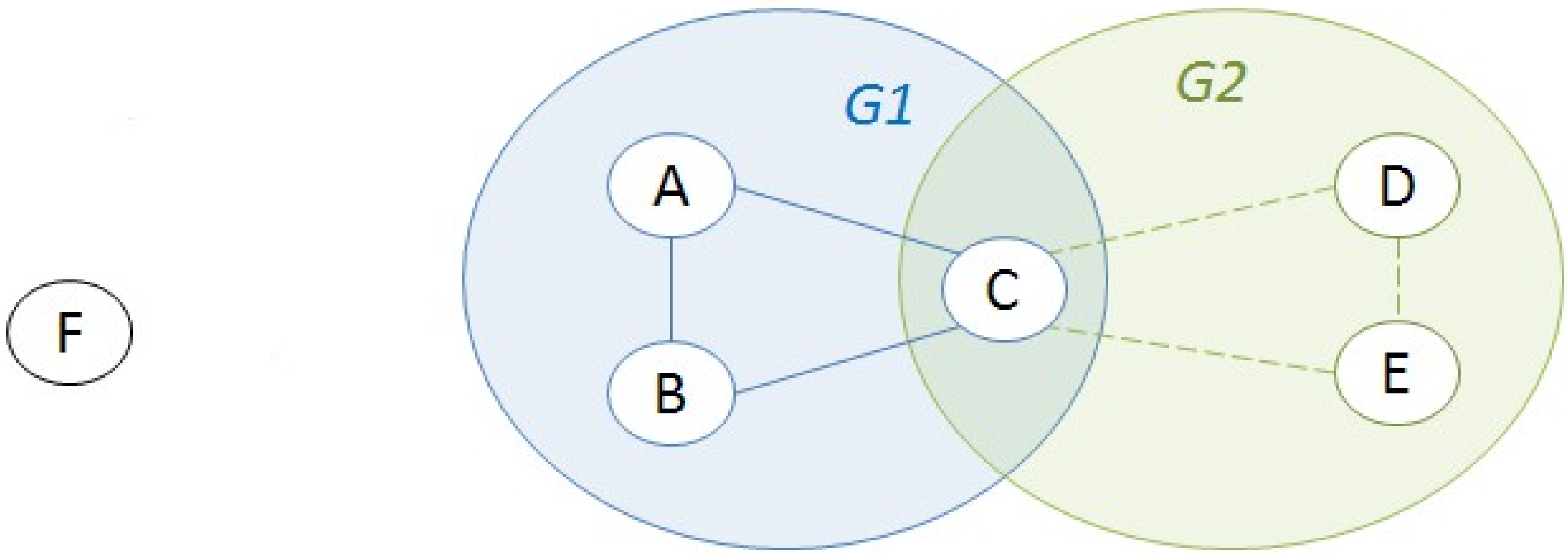
a)



b)

Overlapping Groups & Non-transitive Connections

C: new_s_group(G2, [C, D, E]).

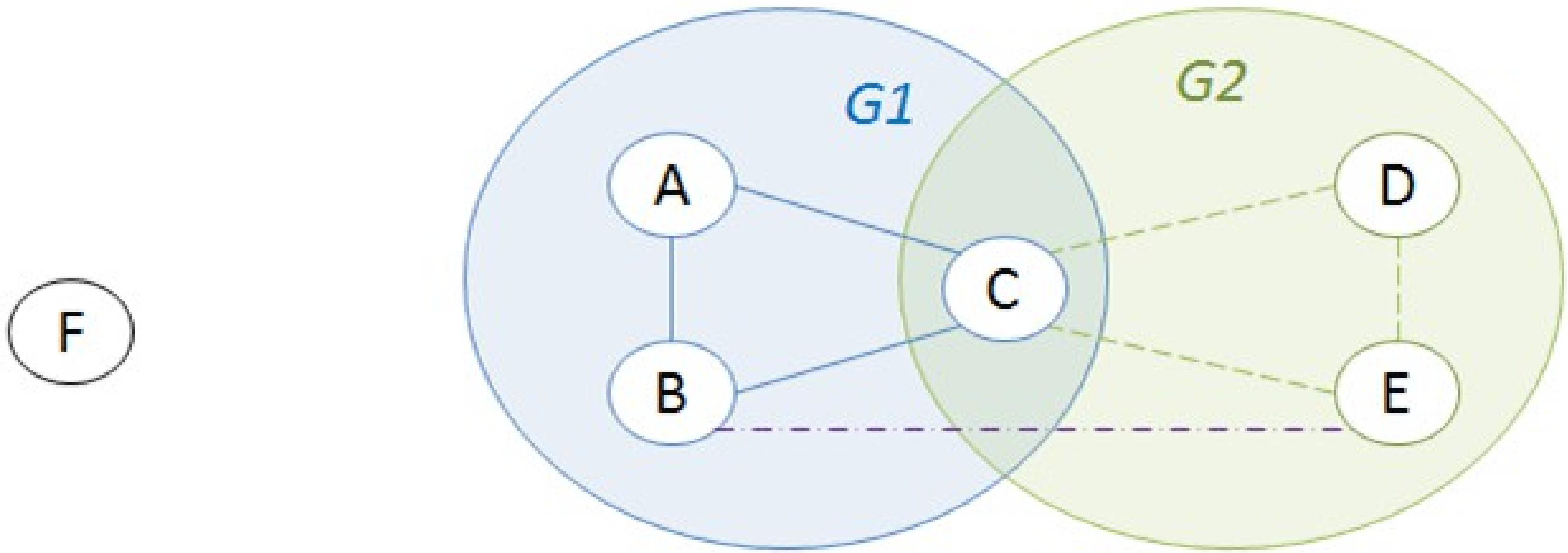


c)

RELEASE

Any to Any Connection

B: spawn(E, *f*).



d)

s_group Primitives

- Creating a new s_group

  new_s_group(S_GroupName, [Node]) -> true | *{error, ErrorMsg}*

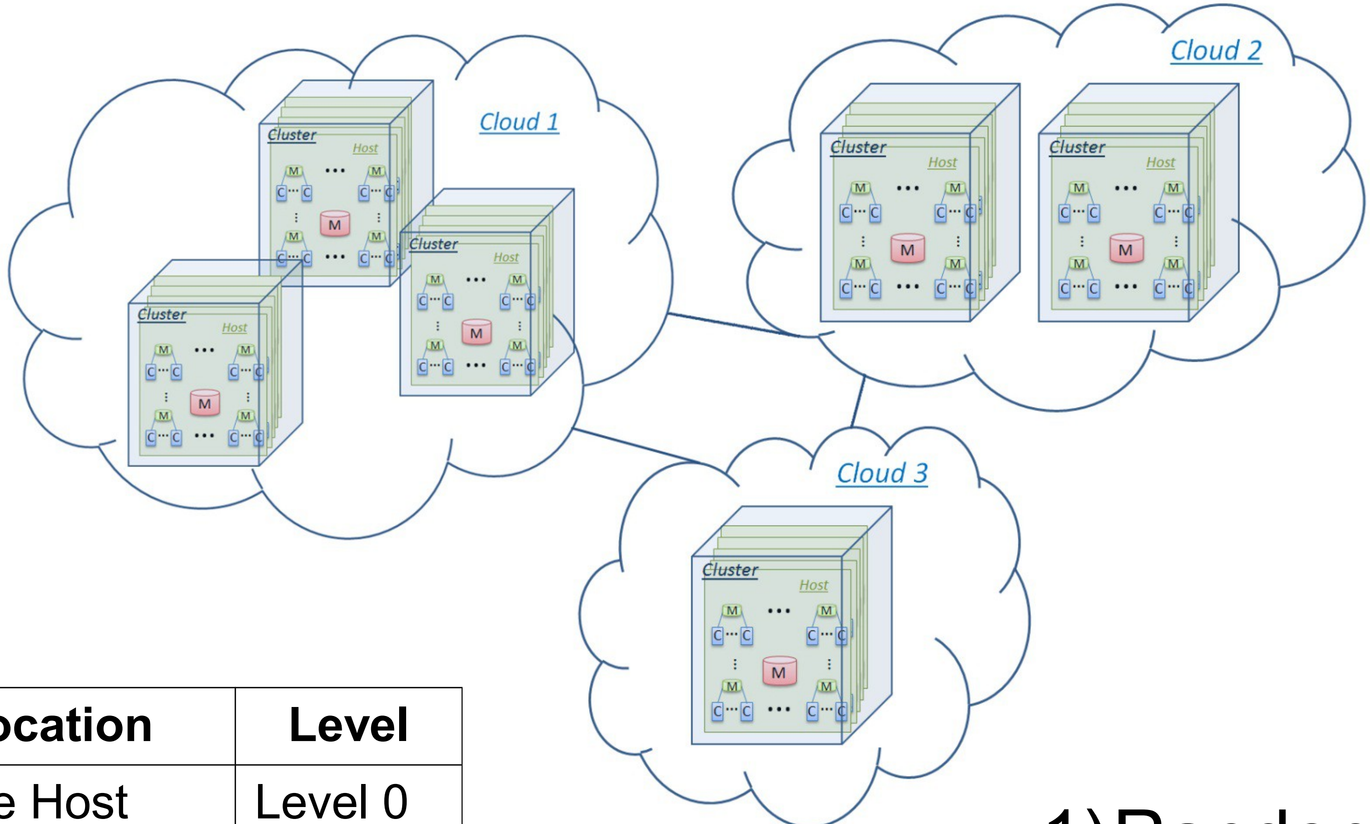
- Deleting an s_group
- Adding new nodes to an existing s_group
- Removing nodes from an existing s_group
- Monitoring all nodes of an s_group
- Sending a message to all nodes of an s_group
- Listing nodes of a particular s_group
- Listing s_groups that a particular node belongs to
- Connecting to an s_group
- Disconnecting from an s_group

s_group Abstractions

- Algorithm skeletons
- Behaviour abstractions
  - s_group supervision
  - s_group master/slave

- We expect the behaviours to become apparent during the work on the case studies and scalable infrastructure.

Semi-explicit placement



Location	Level
Same Host	Level 0
Same Cluster	Level 1
Same Cloud	Level 2
Another Cloud	Level 3

1) Random
2) Load Balancing
3) …

RELEASE

chose_node/1

chose_node(Restrictions) -> node()

Restrictions = [Restriction]

Restriction = *{s_group, S_Group}*

| *{min_dist, MinDist :: integer() >= 0}*

| *{max_dist, MaxDist :: integer() >= 0}*

| *{ideal_dist, IdealDist :: integer() >= 0}*

start() ->
TargetNode = chose_node([*{s_group, G1}, {ideal_dist,* Level0*}]),*

spawn(TargetNode, fun() -> loop() end).

RELEASE

Conclusion

- We have presented an SD Erlang design
  - S_groups
    - Transitive intra group connections
    - Non-transitive (short lived) inter group connections
  - Semi-explicit placement
- We are implementing it now

Thank you!

RELEASE

Exemplar Summary

No	Property	Sim-Diasca	Orbit	Mandelbrot	Moebius	Riak
			S_groups			
1	**Static/Dynamic**	Static	Static	Static	Dynamic	Dynamic
2	**Grouping**	Locality	Hash table	Locality	Multiple	Preference list
3	**Custom Types**	Yes	No	No	Yes	No
			General			
4	**Num. of nodes and s_groups**	Ng << Nn	Ng << Nn	Ng << Nn	Ng << Nn	Ng >= Nn
5	**Short lived connections**	Yes	Yes	No	No	Yes
6	**Semi-explicit placement**	Yes	No	Yes	No	No

RELEASE