University of Glasgow

RELEASE

Scalable Distributed Erlang

Natalia Chechina
and RELEASE Team

December 4, 2013

University of Kent

HERIOT WATT UNIVERSITY

EDF

ERICSSON

ICCS

UPPSALA UNIVERSITET

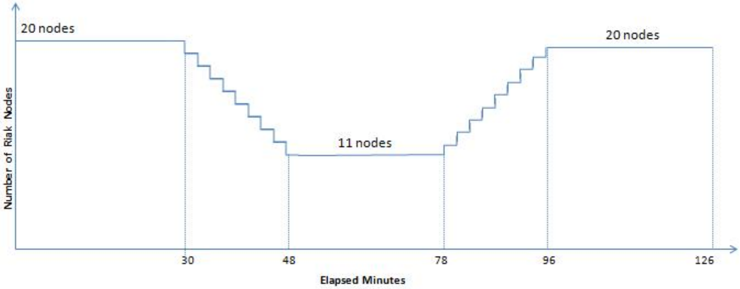
Erlang SOLUTIONS

Outline

1. Distributed Erlang
2. Scalable Distributed Erlang (SD Erlang)
3. SD Erlang Orbit
4. Semi-Explicit Placement

Why Distributed Erlang?

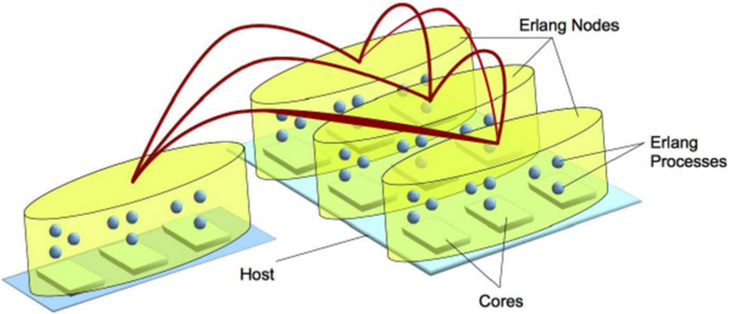
- **Reliability:** multiple hardware and software redundancy means that if one Host or Node fails, other Nodes can continue to deliver service
- **Scalability:** can only scale to around 100 cores on one Host (Node). Many systems use 1000s or 10000 cores



Distributed Erlang

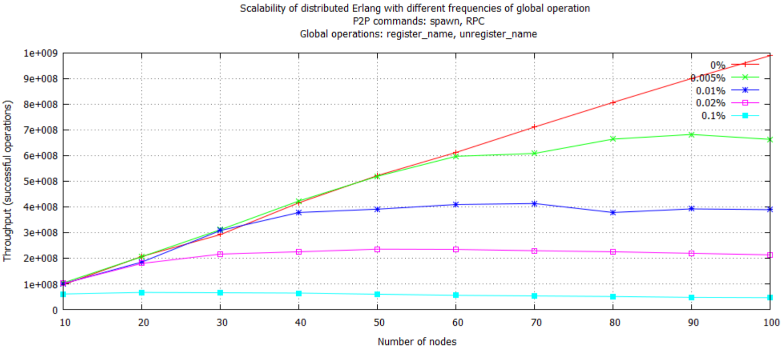
- Transitive connections
- Explicit Placement, i.e.

```
spawn(Node, Module, Function, Args) → pid()
```



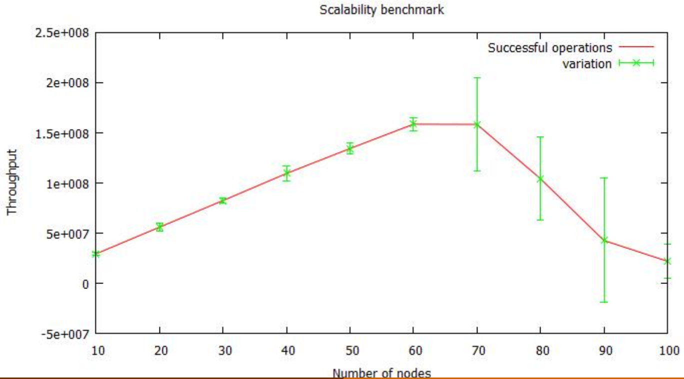
Distributed Erlang Scalability Limitations (1)

- Global operations, i.e. registering names using `global` module
- Other global operations, e.g. using `rpc:call` to call multiple nodes



Distributed Erlang Scalability Limitations (2)

- Single process bottlenecks, e.g. overloading `gen_server`'s `rec` process
- All-to-all connections (no evidence yet)



Why Orbit[LN01]?

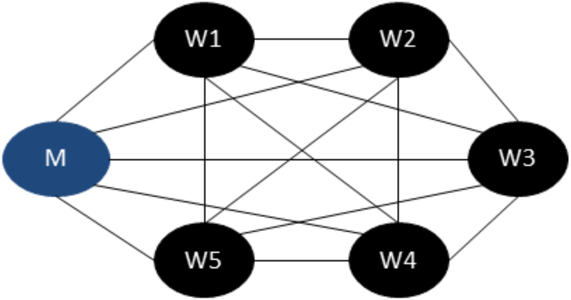
- Uses a Distributed Hash Table (DHT) similar to NoSQL DBMSs like Riak [Bas13], i.e. the hash of a value defined where the value should be stored
- Uses standard P2P techniques and credit/recovery distributed termination detection algorithm [MC98]
- Is only a few hundred lines and has a good performance and extensibility

Orbit in Distributed Erlang

Main components: `master.erl`, `worker.erl`, `table.erl`, `credit.erl`

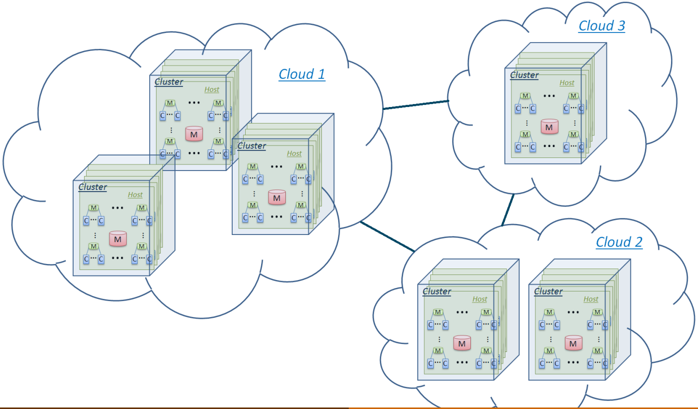
× Pid = spawn_link(worker, init, [TabSize, TmOut, SpawnImgComp])

✓ Pid = spawn_link(Node, worker, init, [TabSize, TmOut, SpawnImgComp])



Typical Target Architecture - 10^5 cores

- Commodity hardware
- Non-uniform communication
  (Level0 – same host, Level1 – same cluster, etc)

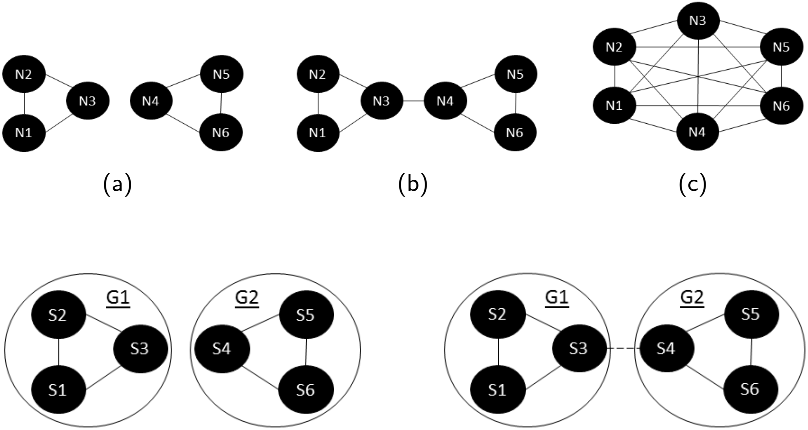


SD Erlang Overview

SD Erlang is a small conservative extension of Distributed Erlang

1. **Network Scalability**
   - Types of nodes
     - Free nodes (normal or hidden) belong to *no s_group*
     - S_group nodes belong to *at least one s_group*
   - Nodes in an s_group have transitive connections only with nodes from the same s_groups, but non-transitive connections with other nodes
2. **Semi-Explicit Placement**

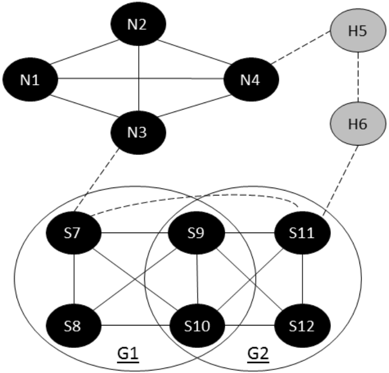
Free Node Connections vs. S_group Node Connections



(a) (b) (c)

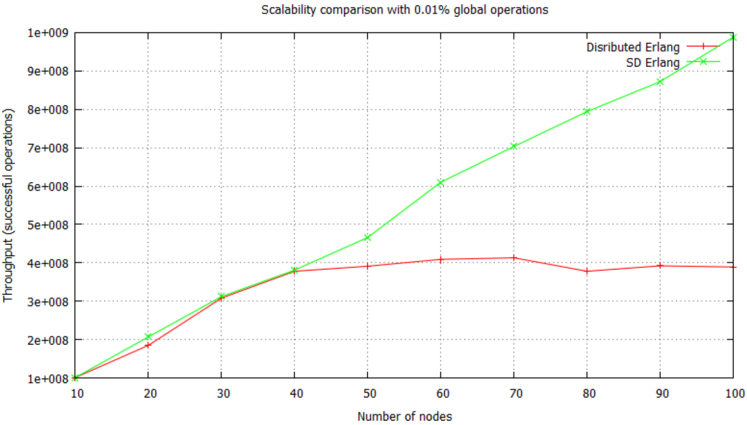
(d) (e)

Types of Connections between Different Types of Nodes



Transitive connection ———
Non-transitive connection --------

SD Erlang Improves Scalability



S_group Functions

s_group:new_s_group/1,2

```
new_s_group([Node]) → {SGName, Nodes} | {error, Reason}
new_s_group(SGName, [Node]) → {SGName, Nodes} | {error, Reason}
```

s_group:delete_s_group/1

```
delete_s_group(SGName) → 'ok' | {error, Reason}
```

s_group:add_nodes/2

```
add_nodes(SGName, Nodes) → {ok, SGName, Nodes} | {error, Reason}
```

s_group:remove_nodes/2

```
remove_nodes(SGName, Nodes) → 'ok' | {error, Reason}
```

Additional SD Erlang Functions

S_group Information

`s_groups/0, own_nodes/0, own_nodes/1, own_s_groups/0, info/0`

Name Registration

`register_name/3, unregister_name/2, re_register_name/3`

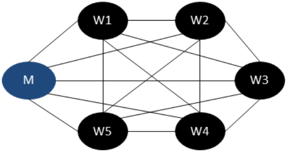
Searching and Listing Names

`registered_names/1, whereis_name/2, whereis_name/3`

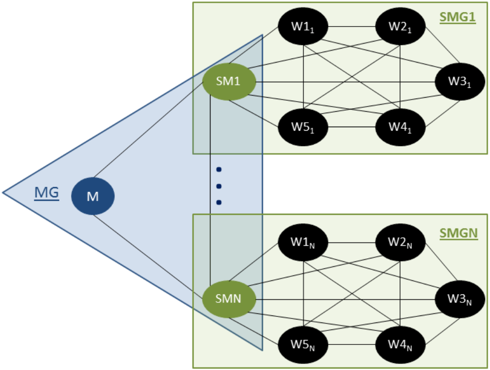
Sending Message to a Process

`send/3, send/4`

Distributed Erlang Orbit vs. SD Erlang Orbit



(f)



(g)

Distributed Erlang Orbit → SD Erlang Orbit

Distributed Erlang Orbit:

- `master.erl, worker.erl, table.erl, credit.erl`

SD Erlang Orbit:

- `master.erl, worker.erl, table.erl, credit.erl`
- + `submaster.erl, grouping.erl`

Details of the differences between the files can be checked by using, for example, `diff module1 module2` unix function

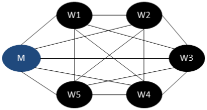
master.erl

Distributed Erlang Orbit

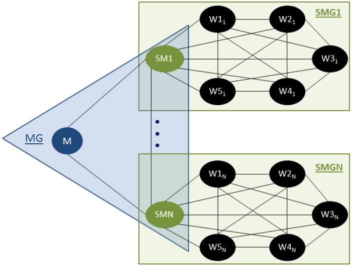
- Spawns worker processes

SD Erlang Orbit

- Spawns submaster and gateway processes



(h)



(i)

SD Erlang master.erl

Distributed Erlang

```
start_workers([{Node, M, TabSize, TmOut, SpawnImgComp} | Hosts],
              {Workers, GTabSize}) ->
...
    Pid = spawn_link(Node, worker, init, [TabSize, TmOut, SpawnImgComp]),
...
```

SD Erlang

```
start_submasters([{Node, GroupName, TabSize} | Sub_masters],
                 {Group_Hash_Tab, GlobalSize},
                 {Gs, Xs, P, Timeout, Spawn, Credit}) ->
...
    Pid = spawn_link(Node, sub_master, init,
                     [Gs, Xs, P, Timeout, Spawn, Credit, self(), GroupName]),
    Gateway = spawn_link(Node, sub_master, gateway, []),
...
```

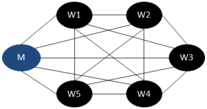
worker.erl

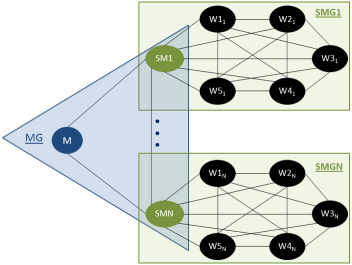
Distributed Erlang Orbit

- Sends a message with vertex X directly to the target process

SD Erlang Orbit

- Sends a message with vertex X directly to the target process only if the process is in the own s_group, otherwise sends it to a gateway process





SD Erlang worker.erl

Distributed Erlang

```
hash_vertex(StaticMachConf, X) ->
...
    %% Translate global slot into worker pid and local slot
    global_to_local_slot(Workers, GlobalSlot).
```

SD Erlang

```
hash_vertex(StaticMachConf, X, K) ->
...
    case (GlobalSlot < Start) orelse (GlobalSlot >= End+TabSize) of
        true ->   %% X should be sent to another s\_group
            Gateway = master:get_gateway(StaticMachConf),
            Gateway ! {X, K},
            ok;
        _Else ->   %% translate global slot into worker pid and local slot
            global_to_local_slot(Workers, GlobalSlot)
    end.
```

SD Erlang submaster.erl (1)

- Initiates submaster and gateway processes
- Submaster processes start worker processes

```
start_workers([{Node, TabSize}|Hosts], {Workers, GTabSize}) ->
    Pid = spawn_link(Node, worker, init, [TabSize]),
...
```

- Submaster processes transfer credit from Worker processes to the Master Process

```
collect_credit(MasterID) ->
    receive
        {done, Credit} ->
            MasterID ! {done, Credit},
            collect_credit(MasterID);
    ...
    end.
```

SD Erlang submaster.erl (2)

- Gateway processes receive {Vertex, Credit} pair and identify its corresponding s_group

```
do_gateway(Group_Hash_Table, StaticMachConf) ->
    receive
        {X, K} ->
            ...
            Gateways = find_gateway(Group_Hash_Table, GlobSlot),
            case Gateways of
                not_found_appropriate_gateway ->
                    throw("not_found_appropriate_gateway");
                _Else ->
                    case lists:member(self(), Gateways) of
                        true  ->
                            %% X belongs to the current s_group
                            %% Forward to the own Worker process
                        _Else ->
                            %% X belongs to another s_group
                            %% Forward to another s_group Gateway
                    end
            end,
...
```

SD Erlang grouping.erl

- Creation of s_groups on Submaster nodes

```erlang
make_group([Submaster|Workers], Counter) ->
    spawn(Submaster, grouping, create_group,
          [self(), [Submaster|Workers], Counter]),
    receive GroupName -> {Submaster, GroupName} end.

create_group(Master, Nodes, Counter) ->
    FixedName = group,
    GroupName = list_to_atom(atom_to_list(FixedName)++integer_to_list(Counter)),
    try
        {ok, GroupName, _Nodes} = s_group:new_s_group(GroupName, Nodes),
        Master ! GroupName
    catch
        _:_ -> sderlang_is_not_installed
    end.
```

- Creation of the master s_group, i.e.

```erlang
    s_group:new_s_group(master_group, [MasterNode|SubmasterNodes]),
```

Semi-Explicit Placement Functions

choose_nodes/1

```
s_group:choose_nodes([Parameter]) -> [Node]
where
     Parameter = {s_group, SGroupName} | {attribute, AttributeName}
     SGroupName = group_name()
     AttributeName = term()
```

Attribute Functions

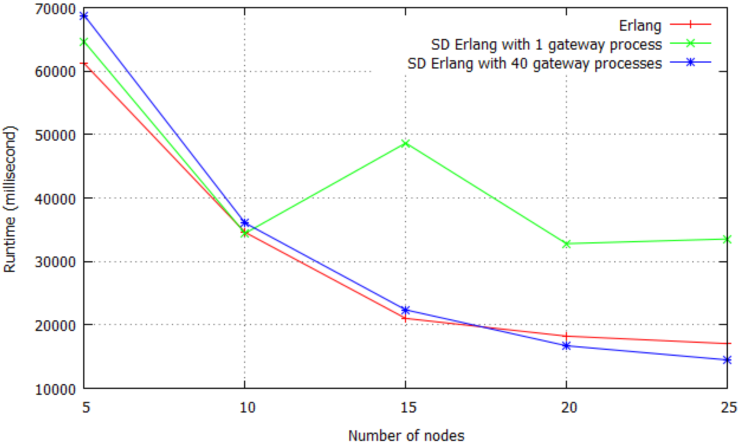
```
global:add_attribute([AttributeName]) -> 'ok' | {error, Reason}
s_group:add_attribute([Node], [AttributeName]) -> 'ok' | {error, Reason}

global:remove_attribute([AttributeName]) -> 'ok'
s_group:remove_attribute([Node], [AttributeName]) -> 'ok'

global:registered_attributes() -> [AttributeName]
```

Thank you!

Benchmarking SD Erlang Orbit on Heriot-Watt University Beowulf Cluster



- BashoConcepts.
  Concepts, 2013.

- Frank Lubeck and Max Neunhoffer.
  Enumerating Large Orbits and Direct Condensation.
  Experimental Mathematics, pages 197–205, 2001.

- Jeff Motocha and Tracy Camp.
  A taxonomy of distributed termination detection algorithms.
  The Journal of Systems and Software, pages 207–221, 1998.