SD Erlang Version of Orbit

Amir Ghaffari, Natalia Chechina, Phil Trinder
The School of Computing Science
Glasgow University
Glasgow, G12 8QQ, UK

March 2014

1 Orbit Calculations

An *Orbit* is a symbolic computing kernel and is a generalization of a transitive closure computation[1]. To compute *Orbit* for a given space $[0..X]$ a list of generators $g1, g2, ..., gn$ are applied on an initial vertex $x_0 \in [0..X]$. This will create new numbers $(x_1...x_n) \in [0..X]$. Repeatedly, generator functions are applied on each of the new numbers and this continues until no new number is generated.

2 Distributed Erlang Orbit

Orbit computation is initiated by a master process. Master establishes a P2P network of worker processes on available nodes. Each worker process owns part of a distributed hash table. A hash function is applied on a generated number to find to which worker it belongs.

To detect the termination of Orbit computation, a credit/recovery distributed algorithm is used [2]. Initially the master process has a specific amount of credit. Each active process holds a portion of the credit and when a process becomes passive, i.e. the process become inactive for for a specific period of time, it sends the credit it holds to active processes. When master process collects all the credit, it detects that termination has occurred.

Distributed Erlang Orbit has a flat design in which all nodes are fully connected. As shown in Figure 1, master process initiates Orbit computation on all worker nodes and each worker node has connections to the other worker nodes.

The following features in Orbit makes the benchmark a desirable case study for SD Erlang:

- It uses a Distributed Hash Table (DHT) similar to NoSQL DBMS like Riak that use replicated DHTs [3].

- It uses standard P2P techniques and credit/recovery distributed termination detection algorithm.
- It is only a few hundred lines and has a good performance and extensibility.

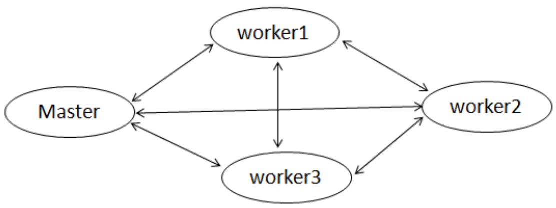

Figure 1: Communication Model in Distributed Erlang Orbit

3 Scalable Distributed Erlang Orbit

Communication Model

To reduce the number of connections, we propose a new design for Orbit in which nodes are grouped into sets of s_groups. In SD-Erlang, s_group nodes have transitive connections with the nodes from the same s_group, and non-transitive connections with other nodes.

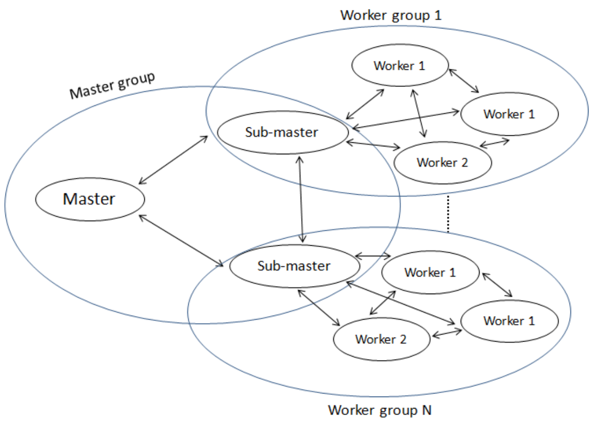

Figure 2: Communication Model in SD Erlang Orbit

As shown in Figure 2, there are two kind of s_groups in this model: `master` and `worker`. There is just one `master` s_group that master node and all the sub-master nodes belong to it. There can be multiple `worker` s_groups in the system. Each `worker` s_group has just one sub-master node and an arbitrary number of worker nodes.

Communication between nodes inside an s_group in done directly but when a worker node needs to communicate with another worker node outside its own s_group, communication is done through sub-master nodes. In this case, the number of messages increases three times, i.e. assume A and C belong to different s_groups, so instead of sending a message from A to C directly (A→C), we would have: A→ $A_{submaster}$ → $C_{submaster}$ →C.

This approach reduces the number of connections between nodes. The number of connections on a worker node is equal to the size of s_group and the number of connections on a sub-master node is equal to the number of worker nodes inside the s_group *plus* the number of all the other sub-master nodes, i.e.

- The number of connections of a worker node is equal to the size of s_group
- The number of connections of a sub-master node is equal to the number of worker nodes inside the group *plus* the number of all the other sub-master nodes

Computation Model

The Orbit computation is started by master process on the master node. Master process dynamically and evenly divides nodes to a number of s_groups. After creating s_groups, the first node of each s_group is chosen as sub-master node. Then, master process runs two kind of processes, i.e. `submaster` and `gateway`, on each sub-master node. A submaster process is responsible to initiate, collect credits and data, and terminate the worker processes in its own s_group and send the collected data back to the master process. In other words, a submaster process is master's representative in an s_group and behaves like an interface between master and worker processes.

The another kind of process on a sub-master node is `gateway`. The number of gateway processes on a sub-master node can be defined based on the number worker process on a worker node. When two worker processes from two different s_groups need to communicate, a gateway processes from each s_groups participate as follow:

Assume `Process1` from `s_group1` needs to send a message to `process2` in `s_group2`. The name of gateway processes are `gateway1` and `gateway2` in `s_group1` and `s_group2` respectively. The message path would be: `Process1`→ `gateway1`→ `gateway2`→ `process2`.

Group1	1-100
Group2	101-200
Group3	201-300

Table 1: Group Hash Partition

Process1	1-25
Process2	26-50
Process3	51-75
Process4	76-100

Table 2: Process Table Partition within Group 1

Double Hashing

In distributed Erlang Orbit, all processes store a copy of hash table that specifies which process is responsible for which fragment of the hash table. However, in SD Erlang design, there are two levels of hash tables. The first level hash table that is created by master process and stored on all sub-master processes, specifies which group is responsible for which part of the table. The second level hash table is created by sub-masters and stored on worker processes. Each worker process stores a table that specifies which process in the s_group is responsible for which part of the table. For example Table 1 shows how range 1-300 is divided among three groups. In Table 2 it is depicted that how Group1 is divided among 4 processes equally.

4 Code structure

This section lists all the modules in SD Erlang Orbit with a brief explanation.

grouping.erl

The grouping module contains functions that dynamically create the s_groups.

- it creates an s_group for the master and all sub-master nodes in function `create_group_list`
- it creates an s_group for a sub-master and all its worker nodes in function `make_group`
- the Orbit space is divided among s_groups in function `create_group_list`

init_bench.erl

Initial value for computing Orbit such as number of processes on each worker node, number of s_groups, and the size of s_groups are set in the main function.

bench.erl

Generator functions and a function for distributed computation are in the module.

worker.erl

All functions related to a worker process such as receiving a vertex, generating new vertexes, and forwarding them to appropriate gateways are in this module.

master.erl

The module creates groups hash table, sub-master processes, gateway processes, and distributes the groups hash table among gateway processes. Collecting credits and the Orbit data also is done in this module.

sub_master.erl

A sub-master process initiates all the worker processes inside its own s_group and collects their credits and data. Gateway process which handles communication between s_groups, is defined in this module as well.

credit.erl

Termination algorithm based on credit/recovery is implemented in this module.

table.erl

The module implemented a hash tables which has a fixed number of slots but each slot can store a list of vertices.

config.erl

Load the configuration file and provides a function for reading the value of a key.

5 Results

This section compares the scalability of SD Erlang and Distributed Erlang Orbits on the Kalkyl cluster. SD Erlang Orbit has been run on the Kalkyl cluster with 11, 22, 44, 66, 88, 110, 132, 154, 176 nodes. The size of s_groups is 11 and in each s_group there are 1 submaster node and 10 worker nodes. For example on 11-node cluster we have one s_group in which we have 1 sub-master node and 10 worker nodes. And, on 33-node cluster, there are

3 s_groups and in each s_group we have one sub-master node and 10 worker nodes, and so in total there are 30 worker nodes and three submasters. There are 40 processes on each worker node, and 60 gateway processes on each sub-master node.

For distributed Erlang version, since there is no sub-master node and all nodes behave as worker nodes, we increase 10 nodes in each experiment. In fact, we compare two versions just with their number of worker nodes. Figures 3 and 4 compare the runtime and the speedup of SD Erlang and Distributed Erlang Orbits. To gain stable results, we run 11 times each experiment and the median value is represented in the diagrams. Vertical lines (the green and pink colors) in Figures 3 and 4 represent 95% confidence interval for the 11 samples. We observe from the runtime comparison:

- The SD-Erlang Orbit has greater runtime than the distributed Erlang Orbit on 1 and 10 nodes (8 and 80 cores)
- The SD-Erlang Orbit scales better than the distributed Erlang Orbit, and by 40 nodes (320 cores) appears to outperform it.
- We can be confident that the SD-Erlang Orbit is outperformaing the distributed Erlang Orbit on 140 and 160 nodes (1120 and 1280 cores)

We also see from the relative speedup comparison:

- Distributed Erlang Orbit shows better speedup than SD Erlang Orbit on 1 and 10 nodes (8 and 80 cores)
- The SD-Erlang Orbit speeds up better than the distributed Erlang Orbit beyond 40 nodes (320 cores).
- We can be confident that the SD-Erlang Orbit is speeding up better than the distributed Erlang Orbit on 140 and 160 nodes (1120 and 1280 cores)

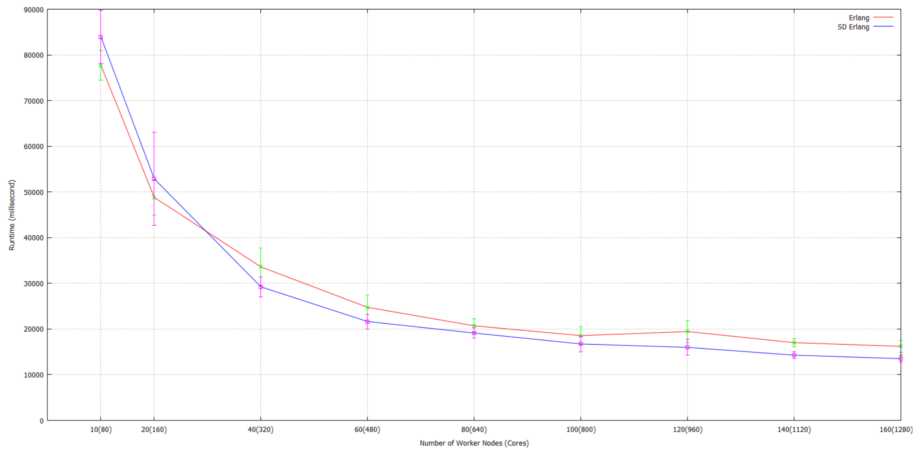

Figure 3: Runtime of SD Erlang and Distributed Erlang Orbits

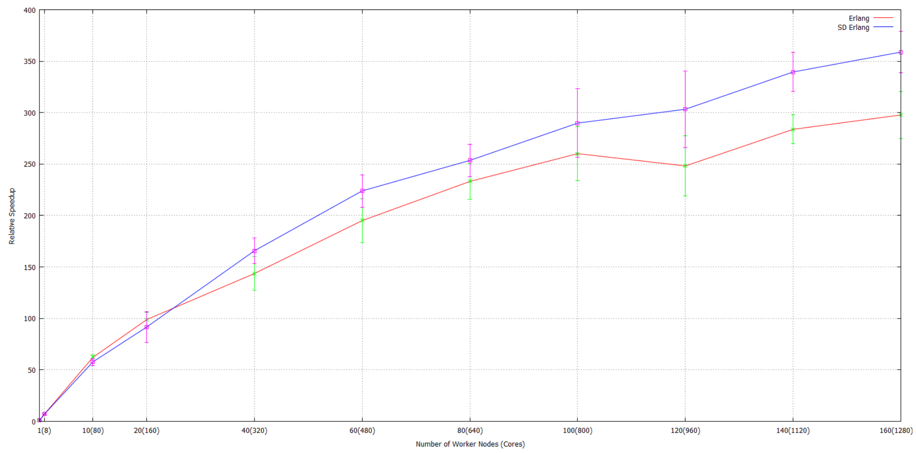

Figure 4: Speedup for SD Erlang and Distributed Erlang Orbits

References

[1] Frank Lubeck and Max Neunhoffer. Enumerating Large Orbits and Direct Condensation. *Experimental Mathematics*, pages 197–205, 2001.

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

[3] BashoConcepts. Concepts, 2013. URL http://docs.basho.com/riak/latest/theory/concepts/.