Inter-domain message passing

Maxim Kharchenko, Cloudozer LLP

08/19/2014

1 Overview

The document describes the message passing within confines of a single physical machine. The message passing in LING is deeply intertwined with all other components of IvanOS. It is (one of) the most fundamental mechanisms of the new operating system.

Erlang mostly hides the distinction between a local and a remote 'Pid'. If needed, it is possible to discern the two using `erlang:node()`. To check whether two Erlang nodes run on the same physical machine is even harder. You have to parse the node name to extract the IP address.

The described schemes makes the physical machine the container runs on an explicitly accessible property. The message passing between physical machines and between containers on the same machine must use different primitives. The document focuses on the inter-domain message passing – exchange of messages between containers on the same physical machine.

The message passing delivers a message (by default, an Erlang term) from the origin to its destination. Both origin and destination are Erlang processes (or equivalent entities).

The described design favours low latency vs high throughput. Bulk not-latency-sensitive transfers should use other mechanisms[1].

2 The address

The standard destination address for message passing contains three parts: machine, container, and process. The address may take a short form when machine and container parts are implied and the long form where machine and container parts are arbitrary. The container part is an integer value that coincides with the Xen domain id. The process part is the serial id of the process inside the container. Both container and process parts are 32-bit. The machine part is a 64-bit value that uniquely identifies a machine within a cluster.

The standard address identifies a process in the IvanOS cluster and allows (multi-hop) delivery of messages. Note that IvanOS does not use other types of addresses – all addresses are standard machine/container/process addresses.

3 Where long Pids come from?

Certain 'spawn' functions return long Pids, yet such functions are remote calls to local 'spawn' functions on the remote nodes and local 'spawns' produce only local Pids? How long

[1] Tubes in LING, tubes.pdf

Pids come to be? They are byproducts of the Erlang serialisation magic. The `term_to_binary()` function always packages a Pid with all its parts (machine/container/process). The `binary_to_term()` function in its turn represent a Pid as a short or a long Pid depending on its machine/container ids. Thus the reply to you message may contain Pids and they maybe long Pids. The question remains how to send the first ever message to a container never contacted before.

This is only achievable with registered process/container names. In addition to Pids, it must be possible to send messages to the following pseudo-addresses represented as 2- or 3-tuples.

```
{Machine,Container,Process}
{Container,Process}
```

`Container` may be represented either as a number (Xen domain id) or an atom (Xen domain name). `Process` must be represent as an atom (registered Erlang name). Possible representations for `Machine` are not defined yet.

Thus if you expect that the machine has a container named 'lincx' and it has a process registered as 'strawman' you may safely send the message as:

```
{lincx,strawman} ! Hello
```

In the course of the message exchange thus initiated you may receive long Pids and then use them as a destination for further messages.

4 Strawman

The 'strawman' is the component that manages inter-domain message passing. When a process what to send a message to another container it puts the message to an envelop:

```
{envelope,Address,Message}
```

The `Address` is either a long Pid or a tuple described above. The `Message` is the message contents serialised using `term_to_binary()`.

The strawman retrieves the container id from the address and checks whether where is an active connection – or a 'straw' – with that container. If there no straw, it create one using a standard negotiation using Xenstore. Straws are never deliberately closed. They go away when one of the communicating domains shut down.

5 Technical details

The new BIFs related to inter-domain message passing:

BIF	Notes
ling:container()	Returns id of the current container
ling:container(Pid)	Retrieves the container id from Pid
ling:machine()	Returns id of the current physical node
ling:machine(Pid)	Retrieves the machine id from Pid