Distribution and clustering

Request for comments
v0.1

December 4, 2012

Background

The standard Erlang/OTP distribution layer suffers from a number of shortcoming that hinder its use for Erlang on Xen. One of them is that Erlang/OTP is optimized for moderate cluster sizes. The Erlang on Xen instance foam may have dozens of thousand of nodes. A new clustering layer is needed.

The new clustering layer draws heavily from the brilliant Erlang process model. Processes form supervision trees using links and monitors. Process have group leaders for routing input and output. Nodes in the new clustering layer are treated similarly.

(OBSOLETE) They may be supervised by other nodes using node monitors. Also each node has a "master node". The master node is the default routing destination of internode messages. By default, the node that started a new instance becomes its master node. The master node can be reassigned later, similary to a process's group leader. Nodes may have registered names just like processes.

The existing 9p layer is used for internode communications. The distribution protocol of Erlang/OTP and epmd daemon are not used.

The new clustering layer draws ideas from MPI, AMQP, ZeroMQ, etc.

Goals

The clustering layer does not assume that the underlying communication infrastructure is reliable. Rather, the opposite assumption is made. The same goes with reliability of nodes. They may and will fail from time to time and the clustering layer should be prepared to deal with it.

Self-imposed Constraints

- Do not change or extend semantics of Erlang, the language.

Public and private clouds

Erlang on Xen may be deployed in private as well as public clouds. Public clouds, such as Amazom EC2, impose a set of constraints mostly on the communication infrastructure. For example, Amazon EC2, allows only TCP, UDP, and ICMP traffic.

Private clouds that allow unrestricted access to the privileged domain do not have these constraints.

The clustering layer should be designed with public clouds in mind. When a functionality available only in private clouds is required it should be clearly marked as “private clouds only” and a fallback mechanism should be descrived if the cluster is being deployed in a public cloud.

Lower layers

The clustering layer is implemented on top of 9p layer. The straightforward mapping to 9p primitives is required for all clustering functions. For most functions such mapping is discussed.

The new clustering requires a set of enhancements from 9p layer. 9p layer should have a robust authentication/authorization and support session restarts.

The communication should not be dependent on a particular transport connections. A typical transport connection is a TCP connection. TCP is the only option in the public clouds. TCP connection may go away, for example, when an instance is migrated. Dropped TCP connection should be reestablished without affecting the 9p connection.

External interfaces

The clustering layer includes dynamic node creation. Spawning new computing nodes requires interfacing with a particular cloud infrastructure. The de-facto standard for cloud infrastructure interface is EC2 API. This API is supported by most cloud stack, including the primary testing platform for Erlang on Xen clustering - OpenNebula.

The Model

The unit of computation in Erlang is a “process”. Each process is identified by a unique Pid.

A process is executed within confines of a container called a “node”. A unique node identifier is obtained trivially given a Pid. A process cannot be migrated to another node. A node can execute many thousands processes concurrently.

Each node runs on a physical server. A physical server can host dozens nodes. Nodes can be migrated between them. The id of the node must survive migration.

Nodes are attached to a network. Each node has a unique network address that survives migration between physical servers.

Each physical server belongs to a "data center". Network connections within a single data center are more reliable and of much lower latency than connections between data centers.

Each node belongs to a single "node group". Only nodes that belong to the same node group may exchange messages. Communication between processes that belong to different node groups may be implemented by higher-level mechanism, such as persistent queues.

Everything is unreliable

All elements of the model are unreliable. The following three types of failures are possible:

Node crashes/hangs. A node crashes due to a software error. Usually, the condition can be readily detected and handled without a delay. Sometimes, a node may "hang", for example, by entering an infinite loop, and the detection of the condition becomes more complicated. This type of errors are expected to gradually become less frequent. $MTBF = 100 days$.

Physical server crashes. All nodes currently running on the affected physical node go away. The condition may be detected by appropriate event may be generated. $MTBF = 500 days$.

Connectity lost temporarily. Nodes on a physical server lose ability to communicate with other nodes. The condition may be detected and reported by the network. The connectivity is restored within 100s. $MTBF = 500 days$.

The model assumes that all failures follow Poisson distribution.

The two clustering functions

The clustering layer should provide the following two primary functions: *internode message passing* – the ability to send messages to processes on other nodes, and *remote process and node monitoring* – the ability to generate events when a process or an entire node goes away.

The clustering layer relies on the underlying layers, such as 9p connections, and intergration with other systems to implement the functions. All other functions needed for running a distributed application are built on top of these two primary functions.

All expectations of the clustering layer with respect to underlying layers and the external systems should be carefully documented.

Nodes talk 9p

All communication between nodes happens over 9p-connections. Such connections are opened between each pair of nodes that need to exchange process messages or other information related to clustering. 9p-connections between nodes may be closed after a certain period of inactivity.

The 9p layer uses a generic transport interface [1]. If a transport connection underneath a 9p-connection is lost, the 9p layer must attempt to reestablish the transport connection. The simplest kind is a transport connection is implemented by 9p_tcp module.

A separate transport for communication between data centers may be added. The inter-datacenter transport may bundle multiple 9p-messages together, have a special behaviour with respect to timeouts, etc.

The transport connection should not provide redundancy such as multiple IP addresses or port numbers. The redundancy is introduced in a predictable way by mounting several service instances at the same point in the local hierarchy.

The only failure that may bubble up from 9p layer is a “mount loss”. It happens when all connections supplying a given mounting point drop and can not be reestablished. Local processes may subscribe to the mount loss event indicating the subtree they are interested in. The subscribed processes receive a message when the mount loss event happens. The subscriptions use the standard monitor()/demonitor() interface with the Type set to “9p”.

Birds of feather

Each node belongs to a single node group. A node group constitute a communication domain for internode message passing. The name of the node group – an atom – can be retrieved by calling erlang:node_group().

When a new node is started the configuration data passed to it contains inter alia the node group and discovery service specifications. Thus all nodes of a given group share a discover service.

The node group is passed in the *uname* field of 9p-attach requests. If the node group in the request does not coincide with the local node group the request is rejected. This mechanism is similar to the magic cookie of Erlang/OTP. This is the only authorization/authentication scheme used by the new clustering layer.

TODO: how new node groups are created?

How nodes find each other

Each node has a globally unique identifier represented as an atom, such as '96f3c8f3-62a1-42d9-85f1-bf8b230e5c39'. The identifier is assigned by hypervisore upon the startup of a new node. The identifier is retrieved from xenstore at /local/domain/<domid>/vm.

In addition the Xen node has a short name that must be unique for a given physical node. It is not possible to make this short name unique across the entire cluster. The short name can be optionally exposed to Erlang application but it is not used when addressing nodes.

If no active 9p-connection exists between the nodes, then the clustering layer should consult a discovery service to retrieve the transport parameters.

All nodes of a node group share the discovery service. The service is accessed via 9p by mounting `/disc`. The service may be mounted multiple times for redundancy. Each instance of the discovery service knows transport parameters of nodes of a given node group only. A loss of the `/disc` mount constitutes a general "loss of connectivity" failure. As a reaction to the loss of connectivity, the node usually shuts down all functionality except a small post mortem process that reports the situation if the connectivity is restored.

Well-known services

9p server embedded in Erlang on Xen can only export top-level directories that contain synthetic files. Each directory implemented by an Erlang module with a defined interface [1]. The list of well-known clustering services that can be exported by a node:

/proc

The synthetic directory contains a list of files that numeric names corresponding to ranks of processes running on the node. For instance, the file `/proc/137` corresponds to the process with the *Pid* of `<0.137.>`. Writing to the such file is mapped to sending message to it. Each 9p-write sends a single message to the process. The message body must be converted to a binary using erlang:term_to_binary(). Exported by all clustered nodes.

/rproc

The same as `/proc` by files correspond to registered names of processes. For example, the file `/rproc/code_server` allows sending messages to a process registered as code_server. Exported by all clustered nodes.

/disc

A node discovery services. The synthetic directory contains files named after nodes in the communication domain. Reading such files results in a transport configuration needed to reach the node. For example, reading the file `/disc/96f3c8f3-62a1-42d9-85f1-bf8b230e5c39` may yield `{'9p_tcp',{{10,0,0,111},564}}`, the transport configuration needed to connect to *NodeId* of '96f3c8f3-62a1-42d9-85f1-bf8b230e5c39'. Exported only by discovery servers.

How messages travel between nodes

Addressing scheme

Let's assume a message *Msg* is being sent to a remote process *Pid*. The remote process runs on the node *NodeId* and has a local address of *LocAddr* there. *LocAddr* is sufficient to complete the delivery once the message reaches *NodeId*. *LocAddr* is either a rank of the process or a registered name. *NodeId* and *LocId* can be trivially derived from *Pid*.

The addressing scheme may be extended by allowing registered names for nodes. Even further, multiple nodes may have the same registered name and a message addressed to a (registered) process on a named node may reach the process of one of them. If needed, this function can be implemented on top of the simple remote message passing.

Each node participating in the internode message exchange must export two synthetic filesystem `/proc` and `/reg_proc`. Writing to synthetic files `/proc/<rank>` constitutes sending messages to corresponding processes. If *LocId* is a registerd name (atom), then messages should be written to `/reg_proc/<name>`. The message is serialized using the standard erlang:term_to_binary() function. The framing is provided by the 9p layer. A single call to 9p-write corresponds to a single message sent. Note that the maximum message size should be set high to allow large message through.

Connection exists

Lookup transport - node service

Open connection

Possible failures

Starting new nodes

EC2, get parameter list, provide the mapping

How /disc service is updated, bulk updates, sync

Links and process monitors

TODO

A node goes down

TODO

References

[1] 9p-based filesystems, v1.2, Cloudozer, 2012.