1 Super-peer Node

2 Peer Node

Proposed

Hybrid structures are notably deployed in collaborative distributed systems. The main issue in many of these systems is to first get started, for which often a traditional client-server scheme is deployed. Once a node has joined the system, it can use a fully decentralized scheme for collaboration.

Relating to a specified function choreography X belonging to resource owner R, a peer P of our system can be in one of the following states:

Active State When P has been marked as responsible for manage all invocation requests of X forwarded by end users.

Forwarder State Otherwise

function choreographies (FCs) or workflows of functions.

As known, in server-less computing platforms, computation is done in **function instances**. These instances are completely managed by the server-less computing platform provider (SSP) and act as tiny servers where a function is been executed.

3 Resources

4 System's resources and actors

4.1 resource owner

A *resource owner*, henceforward denoted with R, represents an entity capable of *creating*, *modifying* and *authorizing* access to several resources of our system.

Given a resource owner R, there are two type of resources which he can manage:

1. Function choreographies.
2. Server-less function implementations.

4.2 Function choreographies

An *end-user* represents, instead, a third-party application that wants execute one or more function choreographies.

5 Server-less function set

6 Function Choreography Scheduler

First of all, let R a resource owner and R_X a set of server-less functions already defined and deployed by R on an *unique* server-less computing platform provider; we assume that $|R_X| \geq 1$.

Lastly, let k the max number of function instances, executable at the same time on the server-less computing platform provider, which are available to execute any function x_j, where $1 \leq j \leq n$, belonging to R_X

Then, it is said that a *server-less scheduler* $S_{(R_X,m)}$ represents a queuing system, implementing any scheduling discipline, equipped with m so-called *virtual function instance*, where $m \leq k$, which aim is to perform scheduling activity managing any function belonging to R_X. The parameter m is also called *scheduler capacity*.

A virtual function instance represents a function instances on the server-less computing platform provider which is *virtually* owned by that scheduler. In other word, m represents the max number of server-less functions, belonging to R_X, which a given scheduler $S_{(R_X,m)}$ can effectively invoke and perform using the server-less computing platform provider.

6.1 Proprieties and constrains regarding *scheduler capacity*

According to proposed solution, a scheduler capable to manage any function belonging to R_X, if exist, is *not* unique, although it is unique inside a peer node.

In order to achieve better performance in terms of network delay experienced by end users, fault tolerance and load balance, any peer nodes can hold a $S_{(R_X,m)}$ scheduler in order to manage incoming request sent by several users spread in different geographic regions.

However, despite there is no upper bound to the number of schedulers of type $S_{(R_X,m)}$ existing at the same time in our system, there is a limitation regarding the scheduler capacity of each existing scheduler.

To be more precise, let's say that, globally, a there are a set of schedulers $S_{1,(R_X,m_1)}, \ldots, S_{p,(R_X,m_p)}$ exist at the same time, where $p \in \mathbb{N}$ with $p \geq 1$. Following constrains must be hold:

$$\sum_{i=1}^{p} m_i \leq k \tag{1}$$

In other words, the sum of all scheduler capacities must be less or equal to the max number of function instances executable at the same time on the server-less computing platform provider for a given R_X set.

Is very important to make clear that different server-less computing platform providers may establish limits, regarding the max number of function instances executable at the same time, in different way; some of them imposed limits per-account, others per-functions. Our design can accommodate both approaches because:

- If a provider imposed limits per-account, then R_X will contains all functions defined and deployed by R, then k will be represent the provider's global limit.
- If a provider imposed limits per-function, then R_X will contains only one function belonging to R, therefore k will be represent the provider's per-function limit.

7 Abstract Server-less Function

An *abstract function* represent an abstract descriptions of a corresponding server-less function implementation.

That description includes:

- d
- f
- d

Any abstract function can be uniquely identified by an ordered pair (a, b), where a is the *resource owner name* while b is the *abstract function name*.

Given an abstract function, a resource owner can provide different implementations which, although they must be semantically equivalent, may expose eventually different performance or cost behaviour.

We call *concrete function* any implementation of a given abstract function and it is uniquely identified by an ordered tuple (a, b, c), where a and b are, like before, the resource owner name and the abstract function name respectively, c represent, instead, the *function type*, which is an abstract descriptions of their corresponding function implementations

8 Function choreography

A **function choreography** is a resource which represents, using graph notation, all paths that might be traversed through a server-less function composition during its execution.

Formally, being R the resource owner, a **function choreography**, denoted as FC_R, is a graph $G(V, E)$, where V is a set of abstract functions connected each other using predefined control-flow operators (`if`, `for`, etc.).

$$V \overset{def}{=} (f_1, \ldots, f_n) \qquad n \in \mathbb{N}, n \geq 1 \tag{2}$$

$$V \overset{def}{=} R_{X_1} \cup \ldots \cup R_{X_n} = \bigcup_{t=1}^{n} R_{X_t} \qquad t \in \mathbb{N}, t \geq 1 \tag{3}$$

In other words, every vertex $v \in V$ represents a server-less function f which belongs to some set R_{X_t} that, in turn, represents a set of server-less functions, belonging to R, which share same server-less computing platform provider, including its limits in term of max number of function instance runnable at the same time.

Cleary, inside a function choreography

V Let $t \in \mathbb{N}$ with $t \geq 1$, a **function choreography** $FC_{(R_{X_1}, \ldots, R_{X_t})}$ is a set of server-less functions sets.

9 fds

9.1 Function Choreography scheduling

Let FC_R a function choreography belonging to a resource owner R, containing a set F of abstract functions. Formally:

$$F = \{f_1, \ldots, f_n\} \qquad n \in \mathbb{N}, n \geq 1 \tag{4}$$

where f_i denote an i-th abstract function, for $1 \leq i \leq n$.

In order to effectively start the execution of a function choreography, is required that for each abstract function $f_i \in F$ *at least one* concrete function which implements it exist; we denote the latter with $f_{i,concrete}$.

More generally, since multiple implementations of a same abstract function can exist at the same time, which can be deployed on different server-less platform providers too, is required that:

$$\begin{array}{ll} \mathbf{R} \overset{def}{=} R_{X_1} \cup \ldots \cup R_{X_s} & s \in \mathbb{N}, s \geq 1 \\ \forall f_i \in F, \quad \exists f_{i,concrete} \in \mathbf{R} & 1 \leq i \leq n \end{array} \tag{5}$$

where R_{X_i} denote the i-th concrete server-less function set, for $1 \leq i \leq s$.

Is very important to remember that every function implementation $r \in R_{X_i}$ share both the same provider and the same limit regarding the max number of function instances executable at the same time on the server-less computing platform provider.

For example, let $r \in R_{X_i}$ and $g \in R_{X_j}$, with $i \neq j$, r and g. Generally, r and g can share the same server-less computing platform provider, but they may also not. Is very important that r and g cannot share the same limit regarding the max number of function instances executable at the same time. Clearly, this design is required to support providers which impose per-function limits, supporting hybrid-scheduling.

9.2 FC_R-Active Peer Node

In order to effectively invoke all server-less concrete function belonging to a function choreography FC_R, we said that a peer node must be a so-called *FC_R-active peer node*, or, simply, *active*.

To become an active node, it must hold all schedulers objects needed to schedule and invoke on server-less platform any possible concrete function implementation of all abstract function belonging to a function choreography. Formally:

$$R_{X_i} \in \mathbf{R}, \qquad \text{the node holds } S_{(R_{X_i}, m_i)} \text{ with } m_i \geqslant 1 \qquad i \in \mathbb{N}, \quad 1 \leq i \leq s, \tag{6}$$

where $S_{(R_{X_i}, m_i)}$ is the scheduler object necessary to schedule and invoke all server-less function belonging to R_{X_i}, while m_i is its capacity factor.

If aforementioned constrained is not hold,

9.3 dasddasasd

object with only one limitation:

Suppose that globally there are a set of schedulers $S_{1,(R_X,m_1)}, \ldots, S_{p,(R_X,m_p)}$, where $p \in \mathbb{N}$ with $p \geq 1$

To be more precise, when a function x_j must to be execute, let s the current number of busy virtual instances, one of the following events may occur:

1. if $s < m$, the scheduler invoke directly the function x_j on the provider.

2. if $s = m$, the scheduler delay the execution of the function x_j on the provider according to implemented scheduling discipline.

Let R a resource owner and R_x its function choreography made up of $R_{x_1}, R_{x_2}, \ldots, R_{x_n}$ unique server-less functions; it is said that a peer node P is **responsible** for R_x when it contains a sequence of schedulers $S_{R_1}, S_{R_2}, \ldots, S_{R_k}$ with $k \leq n$, belonging to R, capable to invoke all server-less function belonging to R_x.

It is said that a

Depending on the definition of the function choreography provided by R and the unique characteristics of back-end server-less providers which execute all serverless functions R_{x_n} of

It is said that a scheduler S is capable to invoke a server-less function when , a scheduler S can invoke multiple

When a peer A, placed "*at the edge*" of the network, receives a new request of invocation for X by an end user, it performs following task in that order:

1. If it responsible It check for it is an already an *active peer* to manage

has found the tracker for a file F, the tracker returns a subset of all the nodes currently involved in downloading F.