# Hive

## Sistema Distribuido para la paralelización de tareas

Carlos David Muñiz Chall
Jesús Manuel Garnica Bonome

## 1. Descripción

El objetivo de este sistema es ejecutar un gran volumen de trabajo usando, las capacidades de paralelización propias a una red de computadoras e interactuar con los clientes como una única entidad. Como prestaciones adicionales se tienen la extensibilidad automatica de la red y una alta tolerancia a fallos basada en la replicación de tareas. El sistema ejecutará cualquier tipo de tarea, simpre que esta cumpla los requisitos definidos en **Uso y definición de tareas**[4] .

Hive **no brinda** proteción contra ataques o uso malicioso de algún componente dado que su diseño supone un entorno amigable.

## 2. Arquitectura

El diseño de componentes y funcionalidades está basado en una arquitectura **superpeer**[1], esta puede ser entendida como un arquitectura **cliente-servidor**, donde el servidor es un sistema distribuido **peer-to-peer**. Los nodos miembros de este subsistema manejan la distribución de las tareas que deben ejecutar los nodos clientes, para organizar la replicación de tareas y el balance de carga, se encuentran enlazados en un anillo, donde cada nodo conoce a su siguiente y su anterior.

En su conjunto el sistema consta de dos tipos de nodos:

- Drone: nodos correspondientes a la parte cliente, en estos es donde se realiza el procesamiento de las tareas.
- Overlord: miembros de la red **peer-to-peer** que constituye el servidor, estos realizan el balance de carga y la replicación de las tareas.

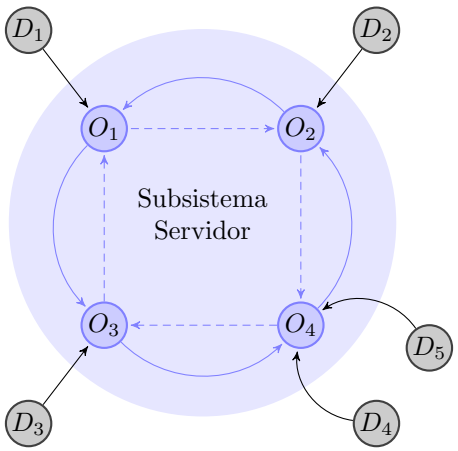


Figura 1: Esquema general de un sistema Hive, Drones = $D_1$,$D_2$,$D_3$,$D_4$,$D_5$ Overlords = $O_1$,$O_2$,$O_3$,$O_4$

### 2.1 Balance de Carga

Cada *overlord* perteneciente al sistema está a cargo de varios *drones*, los cuales solicitan o generan tareas, cada vez que un *drone* se encuentra libre de trabajo, porque termino la tarea asignada o está a la espera del resultado de una subtarea, solicita una nueva tarea al *overlord* asignado.

El balance de carga se logra con dos procesos diferentes, el primero consiste en distribuir lo más equitativamente posible los *drones* entre la población de *overlords*, para esto cada uno mantiene un campo **salto de carga** que es la cantidad de nodos hacia adelante que se pasará al próximo *drone* que solicite un responsable, luego de cada solicitud este campo es aumentado en 1 hasta que llege al tamaño del anillo, momento en el que vuelve a 0.

El otro proceso tiene lugar cuando un *overlord* no puede responder a la solicitud de tareas, porque él mismo no posee ninguna, en este momento se replica la mitad de las tareas pendientes del primer *overlord*, en el sentido normal del anillo, que tenga alguna o se le reponde al *drone* que intente de nuevo luego de esperar un tiempo determinado.

### 2.2 Tolerancia a Fallos

La confiabilidad se basa en la replicación de las tareas, de modo que ninguna tarea este almacenada en un solo nodo del sistema, una caracteristica importante es que tanto el codigo como los datos asociados a una tarea son inmutables, por lo que no es necesario ningún protocolo especial para la consistencia de las replicas.

La replicación ocurre en dos momentos diferentes, al ser recibida la tarea por un *overlord* y cuando este la entrega a los nodos que supervisa. Cuando una tarea es recibida por el *overlord* este lanza una solicitud de replicación dentro del subsistema servidor, con esto se logra que si se pierde el nodo que recibió la tarea, sin que esta halla sido asignada a ningún *drone*, la tarea no se pierda pues las replicas se encuentran disponibles en otros *overlords*. Después de la replicación la tarea es insertada en una cola en los *overlords* que la hallan recibido, la segunda replicación ocurre cuando un *drone* solicita una tarea, la que esta al principio de la cola se le es asignada, pero no es eliminada hasta que no haya sido asignada a un número determinado de *drones*, para evitar que el fallo de un *drone* en conjunto con el fallo de varios *overlords* impida la realización de algún trabajo.

Debido a la replicación la pérdida inesperada de un nodo, sin importar su tipo, no afecta el funcionamiento del sistema, pero hay dos casos que requieren una consideración más profunda; primero, si pierde un *overlord* las tareas no se afectan, pero la estructura de anillo si, en este caso se usan las referencias que tiene cada miembro del anillo hacia el anterior para descubir los dos nodos que deberían estar enlazados al que se perdió, y enlazando estos dos se recupera la estructura nuevamente; segundo, si el nodo perdido es el nodo cliente de la tarea original, no una subtarea generada por esta, el sistema no puede entregar el resultado del cómputo solicitado, pero esto se considera responsabilidad del cliente.

## 3. Protocolo Hive

Para garantizar la extensibilidad del sistema se considera nodo del mismo a cualquier proceso que siga el protocolo que se expondrá a continuación.

Como la mayor parte de dicho protocolo descansa sobre **RPC**[**?**], las acciones se definirán en términos de argumentos y resultados. Para lograr la interacción a través de rpc cada nodo debe mantener un servidor rpc operacional mientras esté activo, la dirección a donde esté sujeto será considerada la dirección del nodo dentro de la red. En este servidor se deben registrar las funciones que se describen para cada tipo de nodo.

Las acciones que puedan demorar más que una simple comfirmación de llegada, aparecen como una pareja de funciones, una en el destino para iniciar la acción y una en el origen para recibir los resultados, minimizando la posibilidad de que un procesamiento lento sea confundido con un error de comunicación.

Las próximas secciones describen el protocolo, separando las acciones por las interfaces de comunicación a la que pertenecen.

### 3.1 Overlord-Overlord

#### 3.1.1 EnterRing

Esta acción es iniciada por un nodo que no pertenece al anillo de *overlords* con el objetivo de incluirse.
**Argumentos:** dirección del nodo que desea unirse.
**Resultados:** (dirección del nodo próximo, nombre para el nuevo nodo del anillo)

El nodo que responde tiene la responsabilidad de informar su nodo próximo que su anterior ya no es él, sino el nuevo nodo, además de modificar sus propias direcciones consistentemente. Quien inicia la acción queda incluido en el anillo con antecesor el nodo al que envió la solicitud y como sucesor el sucesor de este último, si este nodo ya posee un nombre el de la respuesta debe ser ignorado.

#### 3.1.2 SetPredecessor

Informa al nodo receptor que su antecesor ha cambiado.
**Argumentos:** dirección del nodo que será el nuevo antecesor.
**Resultados: ACK**

#### 3.1.3 Address

Informa al receptor el resultado de la solicitud de resolucion de dirección.
**Argumentos:** (a: dirección del nodo solicitado, i: identificador único de solicitud)
**Resultados: ACK**
Esta acción es la forma de recibir los resultados de **Who**[3.1.4], el identificador es único para cada solicitud iniciada por un nodo, no necesita ser único en todo el sistema

#### 3.1.4 Who

Operación de resolución de direcciones dentro del anillo.
**Argumentos:** (c: contador, a: dirección del nodo que inicio la acción **Who**, i: identificador único de solicitud)
**Resultados: ACK**

- $c > 0$: se envia **Who(c - 1, a, i)** al próximo nodo en el sentido normal del anillo.
- $c = 0$: se envia **Address(a, i)**[3.1.3] al nodo con dirección **a**.
- $c < 0$: si el nodo antecesor es accesible se envia **Who(c, a, i)** al nodo antecesor, si este no es accesible se envia **SetPredecessor(dirección del nodo actual)**[3.1.2] al nodo con dirección **a**

#### 3.1.5 Replicate

Ordena la replicación de una tarea dentro del anillo.
**Argumentos:** (t: tarea, sf: factor de salto, sc: contador de saltos, rc: contador de replicaciones)
**Resultados: ACK**

- $sc = 0$: se adiciona **t** a la cola de tareas del *overlord* actual.
  - $rc > 0$: se envia **Replicate(t, sf, sf, rc - 1)** en el sentido normal del anillo.
- $sc > 0$: se envia **Replicate(t, sf, sc - 1, rc)** al siguiente nodo del anillo.

#### 3.1.6 Consume

Fuerza la replicación de tareas hacia un *overlord* que no posee ninguna
**Argumentos:** a: dirección del overlord que inicio la acción **Consume**
**Resultados: ACK**

Si no el nodo no posee ninguna tarea pasa la solicitud al nodo siguiente en el anillo, si él es el que inicio la acción, se espera un tiempo determinado antes de repetirla. En caso de tener tareas pendientes y no ser el nodo que inicio la acción se ejecuta una serie de acciones **PutTask**[3.2.1] hacia el nodo con dirección **a**.

### 3.2 Overlord-Drone

#### 3.2.1 PutTask

Recibe una tarea. Esta acción es implementada por ambos tipos de nodos.
**Argumentos:** t: tarea
**Resultados: ACK**

- *Drone*:
  Esta acción es el resultado de haber invocado **GetTask**[3.2.2] en un *overlord*, la tarea asignada es preparada para ejecutarse cuando el *drone* esté libre de trabajo.
- *Overlord*:
  Puede ser tando el resultado de una acción **Consume**[3.1.6], en cuyo caso se recibirán las replicas cada una en una invocación diferente de **PutTask**, o ser la solicitud de un *drone* cliente de iniciar una nueva tarea o subtarea. La tarea **t** es añadida a la cola de tareas y su replicación es iniciada[3.1.5].

#### 3.2.2 GetTask

Esta acción es implementada solo por los *overlords*, su invocación señaliza que un drone está listo para recibir una nueva tarea.
**Argumentos:** a: dirección del *drone* que solicita la tarea
**Resultados: ACK**

La respuesta a esta acción es enviada a través de **PutTask**[3.2.1], la tarea que se envia es la del principio de la cola, esta tarea es eliminada de la cola depués en un número determinado de invocaciones de **PutTask** que la contengan.

#### 3.2.3 Hello

Inicio del proceso de inscripción de un *drone* en el sistema, aunque es implentada por ambos tipos de nodos tiene significados diferentes, pues la inscripción de un *overlord* en el anillo se realiza mediante **EnterRing**[3.1.1]
**Argumentos:** a: dirección de un nodo en la red
**Resultados: ACK**

- *Drone*:
  Esta es la respuesta de un *overlord* a una acción **Hello**, **a** es la dirección del *overlord* que será responsable del *drone* actual.
- *Overlord*:
  El *overlord* inicia una búsqueda usando **Who**[3.1.4], para encontrar el *overlord* que está a una distancia determinada de él, una vez que llege el resultado de la resolución de dirección esta es comunicada al drone con dirección **a** invocando **Hello** en él. La distancia que debe viajar **Who** aumenta con cada invocación de **Hello** hasta que pase por el nodo que inicio la búsqueda, en este caso la distancia vuelve a empezar en 0.

#### 3.2.4 Register

Segunda etapa del proceso de inscripción de un *drone* en el sistema, implementada únicamente por los *overlords*
**Argumentos:** a: dirección del *drone* a registrar
**Resultados:** (aa: dirección del *overlord* antecesor del que recibe el pedido, ap: dirección del sucesor, n: nombre para el *drone*)

Al recibir la respuesta el *drone* queda vinculado al *overlord* que responde y posee las direcciones del antecesor y sucesor de este como alternativas en el caso de la pérdida del supervisor, si la conexión con los tres *overlords* se pierde, el proceso de inscripción debe repetirse desde el principio. Un *drone* que esté restableciendo su conexión con el subsistema servidor debe ignorar el campo del nombre en la respuesta y utilizar el asiganado la primera vez que se conectó. Cada *overlord* debe garantizar que asigna nombres únicos a cada *drone*, sin importar que este vaya a ser ignorado.

### 3.3 Drone-Drone

#### 3.3.1 TaskResult

Comunica la respuesta de una tarea.
**Argumentos:** (t_id: identificador de la tarea cuya respuesta se está transmitiendo, d: datos de la respuesta)
**Resultados: ACK**

Si el *drone* donde se invoca este procedimiento ya tiene un resultado para la tarea con el identificador transmitido, esta respuesta será ignorada.

### 3.4 Descubrimiento de Nodos

Para comenzar a prestar servicio un nodo debe registrarse en el sistema, esto es conocer la dirección de un *overlord* que esté registrado en el anillo y comenzar con **Hello**[3.2.3] hacia él en el caso de un *drone*, y en el caso de un *overlord* iniciar **EnterRing**[3.1.1].

Un nodo puede obtener la dirección de un *overlord* de dos formas, que el usuario que esta adicionando el proceso al sistema la provea, o realizar una búsqueda automática. Para la segunda se provee de soporte a nivel de mensajes **UDP**.

Cada nodo ya registrado debe mantener un puerto **UDP** respondiendo a cualquier mensaje, con la dirección del *overlord* en el cual está registrado, si es un *drone*, o con su propia dirección si es un *overlord*.

El nodo que desea incorporarse debe utilizar mensajes de **broadcast** en su subred hasta obtener una dirección valida. La obtención automática de esta dirección podría ser imposible, cuando en la subred no existe ningún nodo registrado, en este caso el usuario debe proveer la dirección de algún nodo.

Si un nodo es desconectado de la red por pérdida de los nodos a los que está enlazado, debe iniciar este proceso para descubrir un nuevo punto de enlaze.

El puerto por el cual se efectua el descubrimiento puede variar, para permitir que varios sistemas disjuntos convivan en las mismas subredes.

## 4. Uso y Definición de Tareas

Para tabajar con Hive es necesario proveer el código y los datos de cada tarea. Por tarea se entiende cualquier *scrip* en **Python** que cumpla las siguientes condiciones:

- No importa ningún módulo o paquete que no pertenezca a la biblioteca estandar o sea "tasksupport.py", que es el módulo por el cual se exponen los servicios de control de tareas e importación de dependencias.
- En el caso de que importe "tasksuppor.py", no modifica ninguna variable privada, aquellas cuyo nombre comienze por '_'.
- (opcional) Antes de finalizar devuelve el resultado del cómputo a través de "tasksupport.CurrentTask.finalize(result)", de lo contrario se considera que la tarea finalizó sin ningún resultado, lo cual no es un error
- (opcional) Para importar dependencias utiliza "tasksupport.import(dependence)", para cualquier otra forma de importar no se tienen garantias de que funcione.

Las dependencias pueden ser cualquier archivo valido para importar desde **Python**, estas deben ser proveidas desde la orden de ejecución de una tarea y deben incluir los *scrips* con el codigo de las subtareas, en caso de tenerlas, y las dependencias de las mismas.

## Referencias

[1] Andrew S.Tanenbaum and Maarten Van Steen. *DISTRIBUTED SYSTEMS.* Second edition.