Una aproximación inicial a la integración de las bases de datos en la robótica

Agustín Gallego
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
agustin.gallego@fing.edu.uy

Tutores: Facundo Benavides y Lorena Etcheverry
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
fbenavid@fing.edu.uy - lorenae@fing.edu.uy

***Resumen*—*Robot Operating System (ROS)*[1] es un conjunto de bibliotecas y herramientas diseñadas para facilitar el desarrollo en entornos de programación orientados al control de robots. *Cassandra*[2] es una base de datos no relacional (*NoSQL*) orientada a escalabilidad y alta disponibilidad. En este trabajo, veremos cómo se pueden vincular estas tecnologías para acercar estos dos mundos. El objeto principal de estudio será la integración de las bases de datos, no solo como simples repositorios de datos persistentes, sino como facilitadores de la gestión de los mismos. Además, se verá el rol que pueden cumplir éstas en algunos sistemas robóticos en particular.**

I. Introducción

La robótica y la gestión de datos han evolucionado enormemente como áreas disjuntas, pero al día de hoy, las aplicaciones en las que se tratan como partes de un mismo componente son escasas [1]. Con este trabajo, se tratan de identificar áreas en las que tenga sentido utilizar bases de datos como sistema encargado de la persistencia, organización, y recuperación de datos. De esta manera, futuras aplicaciones dispondrán de una guía inicial en la cual basarse para su uso. Además de identificar las áreas de aplicación en términos generales, se desarrollará un caso de uso en particular, en el cual se demostrará un posible tipo de interacción entre los componentes estudiados. Para ello nos valdremos de una herramienta previamente dessarrollada, e integraremos tres tipos distintos de nodos: un robot (encargado de la generación de datos), un servidor central (encargado de la gestión de datos), y una computadora personal (encargada del postprocesamiento de los datos). La comunicación entre los componentes será a través de internet, para demostrar que no es necesario que exista localidad en este sentido, si bien hay casos en los que pueda ser más conveniente disponer de una red local, como se verá en la sección III-B.

I-A. Objetivos

Uno de los principales objetivos de este trabajo es poder sentar las bases para una discusión más detallada, y brindar herramientas que puedan orientar futuros trabajos en mayor profundidad. Para ello, se planteará un marco teórico en el cual discutiremos la utilidad de las tecnologías relevadas inicialmente, y su posible aplicación en la robótica.

Además, se desea demostrar una aplicación exitosa en la cual se puedan integrar *ROS* y *Cassandra*. Para ello, haremos uso del paquete *cassandra_ros*[3] para modelar un caso de uso de aplicación real: un robot dotado con una cámara, que envía las grabaciones a un servidor central, para luego ser postprocesadas por un centro de mando con mayor capacidad de cómputo. De no disponer de una base de datos central, el robot debería ser capaz de guardar y gestionar estos datos, y de saber cómo y a qué centro de mando enviarlos. Si suponemos que, por el contrario, es el centro de mando el encargado de requerir los datos, surge el problema opuesto: ¿cómo y a qué robot se le deben pedir? En este caso en particular, el beneficio de un repositorio central de datos queda claro, dado que la información necesaria tanto para robots como para centros de mandos es única (y siempre la misma). Aún más, este modelo permitiría fácilmente escalar de un robot y un centro de mandos, a varios robots y varios centros de mando sin mayores consideraciones.

I-B. Organización del documento

El documento está organizado, a grandes razgos, de la siguiente manera:

- Secc. II: Trabajos Relacionados, en los que se mencionan algunos trabajos previos que han sido estudiados y discutidos, y en los cuales nos basamos para este estudio
- Secc. III: Cuepro Central, en el que se introducen las tecnologías estudiadas y los casos de uso relevantes
- Secc. IV: Experimentación, en la que se detallan las pruebas realizadas, y sus resultados
- Secc. V: Conclusión y Trabajos Futuros, en la que se postulan las conclusiones, y se proponen posibles extensiones mediante un listado de trabajos futuros

II. Trabajos relacionados

El principal trabajo detrás de esta investigación es el llevado a cabo por Dietrich et al. en [2], en el cual se plantea la idea original detrás de la integración de *ROS* y *Cassandra*. Los autores han incluso desarrollado una *API (Application*

[1] https://www.ros.org/
[2] http://cassandra.apache.org/
[3] http://wiki.ros.org/cassandra_ros

Programming Interface) que implementa la interacción entre estos dos sistemas (en la Secc. IV se trata en profundidad cómo se utilizó). En el artículo mencionado, los autores desarrollan un caso de uso similar al planteado en la Secc. I-A, con la principal diferencia de que no hay integración con otros sistemas. El robot mismo es el que tiene la base de datos, y en el que se ejecutan los comandos de postprocesamiento. Si bien es una prueba válida, su aplicación está limitada a robots con abundantes recursos disponibles (como ser: capacidad de procesamiento, espacio en disco, memoria, y vida útil de la batería). Como veremos más adelante, las realidades en las cuales puede ser aplicable un proyecto de estas características es más reducida, dado que típicamente los robots cuentan con recursos limitados. Éstos pueden variar desde capacidad de procesamiento hasta espacio en disco, pero ciertamente el común denominador será el uso de una batería como la fuente de energía. Solamente esta limitante nos presenta una cota que no puede ser ignorada, ya que el funcionamiento del robot se ve condicionado enteramente por este recurso.

Otro trabajo relacionado es también de Dietrich et al., en [3]. En este caso, un robot tiene que planificar una ruta dentro de un edificio, utilizando datos previamente guardados en una base de datos. En él, presentan además una jerarquía de ordenamiento de datos, en la cual podemos representar objetos desde lo macro (un edificio), hasta lo micro (un objeto dentro dentro de un cuarto del edificio).

Por último, en [1], Simoens et al. discuten un caso similar, en el que un robot debe planificar una ruta dentro de un edificio hasta una estación de carga. El robot se ayuda de información brindada por cámaras dentro del edificio, para poder detectar más eficientemente dónde se encuentra su objetivo.

III. Bases de datos en sistemas robóticos

III-A. Cassandra

Cassandra es una base de datos no relacional (*NoSQL*[4]) distribuída, que utiliza el modelo clave-valor (*key-value*) para guardar datos. Utiliza familias de columnas (*column families*) ordenadas en espacios de claves (*keyspaces*) para ordenar lógicamente las tuplas (son el equvalente a tablas y esquemas, respectivamente, en los modelos relacionales). La arquitectura está definida por: nodos, la unidad atómica, en donde se guardan los datos; *datacenters*, una colección de nodos física o virtual (típicamente en el mismo centro de datos); y *clusters*, una colección de *datacenters*, que puede abarcar distintas zonas geográficas. Su caso de uso principal son entornos en los que se trabaja con *big data* (datos masivos, o a gran escala), para lo cual se vale de las siguientes herramientas:

- Particionado: una base de datos lógica es dividida en partes iguales por cada nodo. Se utilizan funciones de *hash* para evaluar la clave primaria, y así saber a qué nodo dirigir la lectura/escritura. Esta operación es transparente al usuario, es decir, no importa a qué nodo se efectúa la consulta, los resultados son obtenidos desde el *cluster* por el mismo nodo
- Replicación: para garantizar la alta disponibilidad de los datos, se duplican los datos en *N* nodos (dependiendo del factor de replicación)
- *Hashing* consistente: para evitar operaciones costosas cuando se agregan o remueven nodos, se utiliza un rango de valores asignado a cada uno. De esta manera, solo se precisan reposicionar un conjunto mínimo de tuplas
- Consistencia transaccional: se puede elegir el nivel de consistencia de datos deseado, variando desde local al nodo donde se ejecuta la consulta (no se espera por quórum de las replicas), a global (todas las réplicas deben responder)
- Protocolo de *gossip*: para comunicación entre nodos, cada segundo se intercambian mensajes de estado con hasta tres otros nodos dentro del *cluster*. De esta manera, se acelera el proceso de descubrimiento de nodos caídos
- Tablas ordenadas en disco: cuando los datos en memoria superan una cota, se guardan en las llamadas *SSTables (Sorted String Tables)*, en las cuales las tuplas están ordenadas por clave primaria. De esta forma, la búsqueda en disco puede ser realizada más eficientemente

III-B. Posibles Escenarios

Como mencionamos previamente, hay varios casos en los que la utilización de una base de datos puede ser provechosa. En esta sección detallaremos tres casos que creemos tienen potencial para una investigación en más profundidad.

III-B1. ***Redes de sensores:*** En una red de sensores disponemos de varios nodos que proveen datos sobre, típicamente, condiciones físicas como temperatura o humedad (Mazzara et al. [4]). Estos nodos son generalmente computadoras de recursos extremadamente limitados, debido a que la tarea que deben cumplir es simple, y solo se requiere de ellos el censado de dichos datos; por otra parte, se precisa una cantidad de sensores no trivial, para poder abarcar zonas extensas de trabajo. La red puede ser cableada o inalámbrica, y dentro de las inalámbricas existen a su vez distintos tipos como las mallas (*ad-hoc*) o las redes *ad-hoc* móviles (*MANETs*). Hablaremos más en profundidad sobre este tipo de redes en la Secc. III-D; por ahora basta con entender que existen varios nodos interconectados, que deben proveer datos sobre ciertos parámetros de eventos físicos puntuales.

En particular, pensemos en el siguiente caso de uso. Disponemos de un área de tierra para cultivar, y deseamos utilizar de manera eficiente el recurso agua, regando solamente cuando sea necesario, y además hacerlo en los lugares en los que la tierra esté seca (para algún valor mínimo de humedad dado por el cultivo). Para ello, utilizaremos sensores de humedad en la tierra, que dispondremos a través del área a cultivar. Cada día de riego, el regador (operado por un robot) consultará la base de datos para ver en qué coordenadas hace falta agua, y aproximadamente cuánta. De esta manera podrá planificar los viajes necesarios, y la cantidad de agua por cada viaje. Además, disponer de datos históricos en la base de datos nos permitirá utilizar esa información para poder tomar mejores decisiones a mayor escala, como por ejemplo: “en Enero se

[4]https://academy.datastax.com/planet-cassandra/what-is-nosql

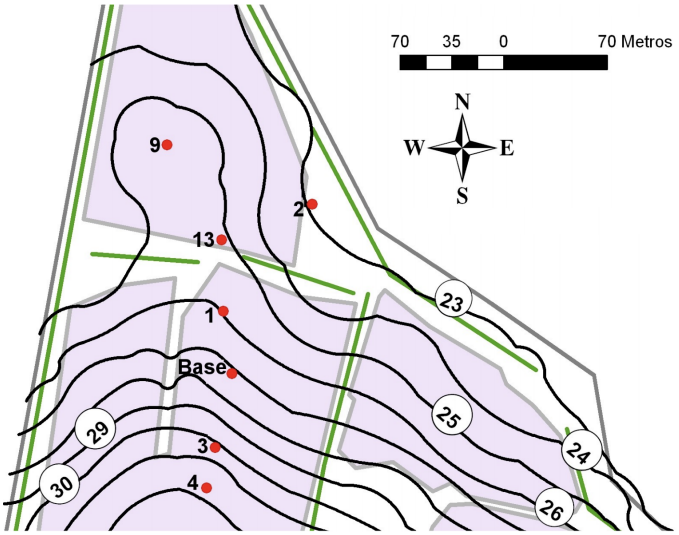

Figura 1: Ubicación de sensores en el predio, del trabajo postulado por Mazzara et al. [4].

precisarán 50m^3 de agua para mantener los cultivos regados al 80 %", o "en los cuadrantes `A`, `B` y `C` se precisa consistentemente un `X` % más de agua que en otros". Por último, y como mencionamos anteriormente, el hecho de utilizar una base de datos centralizada nos permitirá escalar fácilmente las hectáreas que monitoreamos y regamos. En la Fig. 1 se muestra cómo en el trabajo mencionado inicialmente (Mazzara et al.) los autores colocaron los sensores en el predio a censar.

III-B2. ***Soporte en una flota****:* Una flota de robots debe realizar una tarea compleja, subdividiendo el problema en partes más simples. En este escenario dispondremos de dos tipos distintos de nodos. Unos serán los nodos centinela o exploradores, que serán los encargados de llevar a cabo una tarea y reportar los datos resultantes. Los otros serán los nodos de soporte propiamente dichos, encargados de recolectar y gestionar la información generada por los nodos centinela. El beneficio directo en este caso es que los nodos centinela no deben tener toda la información del problema, sino que pueden limitarse a operar con la información parcial que les permita llevar a cabo la subtarea que tienen asignados. De ser necesario, podrán descartar información vieja en pos de información actualizada, una vez deban cambiar de tarea. Los nodos de soporte estarán dotados de *hardware* y *software* especialmente diseñados, para poder cumplir con los requerimientos más demandantes. Para ello, se debe tener en cuenta que estos nodos no solamente se comunicarán con los nodos centinela, sino que se comunicarán con otros nodos de soporte. Algunos ejemplos pueden ser:

- una antena para comunicación con otros nodos de soporte (que provea mayor cobertura o rango de comunicación)
- espacio en disco suficiente para almacenar los datos generados por los nodos centinela
- memoria suficiente para poder manejar un proceso (o servicio) de base de datos
- suficientes núcleos para poder manejar la carga de trabajo deseada como cota superior
- batería con capacidad suficiente para el ciclo de vida esperado

En la Fig. 2 vemos un posible diagrama de comunicación con tres nodos de soporte, y varios nodos centinela conectados a ellos. En este escenario, los nodos centinela pueden intercambiar información con nodos distantes de manera transparente, extendiendo de esta manera el rángo de comunicación entre pares. A su vez, los nodos de soporte podrán estar en contacto con nodos más lejanos, que no pertenezcan a esta topología, y compartiendo información proveniente de ellos.

III-B3. ***Desplazamiento en topologías desconocidas****:* Por último, mencionaremos el caso de un robot de entrega (o retiro) de paquetes. En este escenario, un robot es encargado con la tarea de entregar (o retirar) un paquete en una cierta dirección, la cual incluye número de apartamento, nombre del destinatario, etc. En este caso, el robot no puede valerse de recolectar información del medio, ya sea porque no existe, porque puede ser incorrecta, o por la cantidad de tiempo (o recursos) que esta actividad podría requerir. Al llegar al edificio, el robot accederá a la base de datos para consultar el plano del edificio, y así poder calcular una ruta hacia el destino final deseado. Como beneficio adicional, se podrá compartir información sobre dispositivos que provean datos dinámicos sobre el entorno, como ser cámaras de CCTV (circuito cerrado de televisión), o sensores de proximidad o localización. De esta manera, el robot dispondrá de toda la información necesaria para, no solamente llegar a destino, sino que poder hacerlo de una manera eficiente, esquivando objetos, o descartando caminos bloqueados o muy concurridos. Luego de terminada su tarea, el robot podrá descartar los datos obtenidos para dar lugar a los necesarios para la próxima.

Cabe mencionar que el uso de una base de datos aporta también en el área de gestión de usuarios, y de los niveles de acceso a la información que cada tipo de usuario pueda

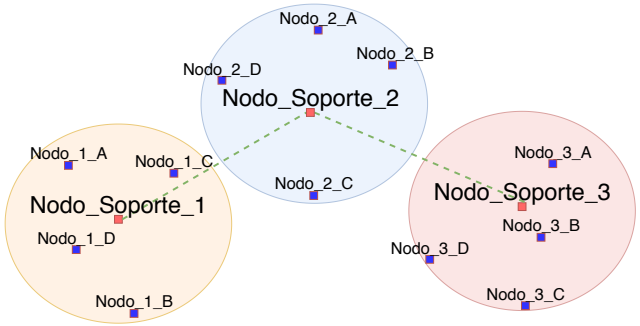

Figura 2: Tres nodos de soporte conectados entre sí, brindando apoyo de comunicación y gestión de datos a los nodos centinela.

llegar a tener. Por ejemplo, un robot de entrega tendrá permiso de lectura sobre ciertos esquemas solamente (aquellos que se crean necesarios son suficientes para llevar a cabo las tareas antedichas), y no tendrá permiso de escritura para ningún esquema. Además, el protocolo de comunicación permite hacer uso de canales confiables (mediante SSL/TLS[5] [6]) para evitar filtración de información a terceros. De ser necesario, se pueden incluso crear usuarios para cada robot en particular (previo acuerdo con las empresas de reparto, por ejemplo) y así evitar que la información sea accedida de manera pública.

III-C. Cassandra_ROS

El paquete *cassandra_ros*[7] fue el principal motor en el desarrollo de los experimentos detallados en la Secc. IV. En él, los autores proveen, mediante una *API*, funcionalidades para la interacción con la base de datos desde el entorno de desarrollo (en *Python*[8]). Se definen varios tipos de mensajes soportados, de los cuales se destacan el "*ros*" (que provee soporte con los mensajes nativos de ROS[9]) y el "*binary*" (que típicamente es más eficiente en cuanto a velocidad de conversión y escritura). La traducción desde y hacia *Cassandra* se hace automáticamente, y es totalmente transparente para el usuario, tomando como clave primaria la fecha y hora de generación del mensaje (*timestamp*). Otra traducción que se hace automáticamente es la del nombre de la familia de columnas a un *hash* (utilizando el algoritmo *MD5*), que puede ser confuso si se consulta directamente en la base de datos. En *Cassandra*, existe una limitación de 48 caracteres para nombres de familias de columnas (y espacios de claves)[10]. Los autores decidieron salvar esta limitación calculando un *hash* a partir del nombre del tópico[11] de ROS del cual se consumen los mensajes.

Para instalar y compilar el proyecto, se utiliza la herramienta *catkin_make* disponible en ROS. Los comandos necesarios para instalar el proyecto y sus dependencias (partiendo de una instalación *Ubuntu 18.04*) se muestran en el Lis. 1.

Listado 1 Instalación de *cassandra_ros* y dependencias

```
$ cd ~/catkin_ws/src
$ git clone https://gitlab.com/OvGU-ESS/cassandra_ros
$ cd .. && catkin_make
$ pip install pycassa cql thrift==0.9.3
$ sudo apt-get install python-qt4 ros-melodic-usb-cam
```

También se provee de un caso de uso funcional mediante archivos de ejecución previamente configurados (bajo el directorio *launch* del proyecto):

- `cassandraBag.launch`: define la conexión con un servicio *Cassandra* configurado en *localhost:9160* por defecto. Es utilizado por el resto de los servicios definidos
- `recordCamera.launch`: se conecta a la base de datos, y guarda la información generada por una cámara conectada vía interfaz *USB*
- `replayCamera.launch`: se conecta a la base de datos, y reproduce un video previamente guardado
- `deleteCamera.launch`: borra un video previamente guardado en la base de datos

Para comunicación con la base de datos mediante la *API*, se provee además un servicio llamado *CassandraBag*[12]. Éste puede ser invocado mediante comandos *ros_run*[13] utilizando la interfaz de línea de comandos *cassandraBag-cli.py*. En el Lis. 2 se muestran ejemplos de comandos posibles.

Listado 2 Ejecutando el servicio *CassandraBag*, y guardando sesenta segundos de video

```
$ roslaunch cassandra_ros cassandraBag.launch
$ rosrun cassandra_ros cassandraBag-cli.py record start \
   /usb_cam/image_raw/compressed
$ sleep 60
$ rosrun cassandra_ros cassandraBag-cli.py record stop \
   /usb_cam/image_raw/compressed
```

Finalmente, es de destacar que el proyecto depende de algunas configuraciones de *Cassandra* que no están dentro de las consideradas "buenas prácticas". Como ser:

- Clave de partición: en la documentación, se insta a utilizar `byteOrderedPartition` como estrategia de particionado de los datos. Esta opción ha sido señalada como no recomendada desde la versión 3.0[14]
- Protocolo de comunicación: se utiliza el protocolo de llamada a procedimiento remoto (*RPC*, por su sigla en inglés -*Remote Procedure Call*-) *Thrift*. Ha sido sustituído por una nueva implementación llamada *CQL (Cassandra Query Language)*[15]

Como corolario del primer punto, se puede ver que en los ejemplos mencionados en [2], los autores utilizaron un entorno con solamente un nodo de base de datos. Como se mencionó en la Secc. III-A, uno de los fuertes de *Cassandra* es la habilidad para hacer que varios nodos trabajen juntos como parte de un *cluster*, garantizado la alta disponibilidad y la tolerancia a particiones en la red. Al hacer uso de solamente un nodo, se están perdiendo todas las garantías en este respecto, y el uso de *Cassandra* pierde gran parte de sus fundamentos teóricos.

III-D. Redes Móviles

Como se vio en los casos de uso planteados en la Secc. III-B, en algunas aplicaciones es provechoso disponer de una malla, o una red móvil en malla (*MANET*, por su sigla

[5]https://www.ssl.com/faqs/faq-what-is-ssl/

[6]https://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/secureSSLClientToNode.html

[7]https://wiki.ros.org/cassandra_ros

[8]https://www.python.org/

[9]http://wiki.ros.org/msg

[10]https://docs.datastax.com/en/cql/3.3/cql/cql_reference/refLimits.html

[11]https://wiki.ros.org/Topics

[12]https://wiki.ros.org/cassandra_ros#CassandraBag

[13]https://wiki.ros.org/rosbash#rosrun

[14]https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archPartitionerBOP.html

[15]https://academy.datastax.com/planet-cassandra/making-the-change-from-thrift-to-cql

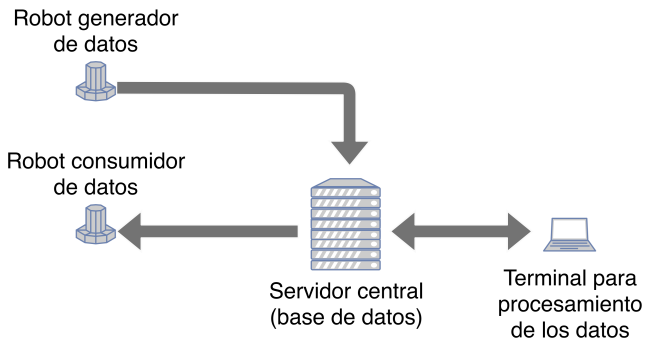

Figura 3: Diagrama básico mostrando una posible interacción entre componentes del entorno de trabajo.

en inglés *Mobile Ad-hoc Network*). La malla permite que cualquier nodo oficie de enrutador de paquetes (nodo de *relay*), extendiendo de esta manera el alcance de nuestra red de sensores o robots [5]. En [6] podemos ver estrategias de posicionamiento dinámico para poder elegir los nodos de *relay* dentro de una topología tipo *MANET*. En el caso de soporte en una flota (Secc. III-B2), no solo los nodos centinela utilizarían una red de este estilo, sino que podría estar presente y coexistir con la red privada entre los nodos de soporte. Se deberá tener la precaución de no saturar la red con paquetes para la comunicación entre los nodos de soporte, limitando la utilidad que esta topología podría brindar. En el caso de redes de sensores (Secc. III-B1), podría ser de utilidad para extender el rango de los sensores, y además brindar apoyo al robot que debe realizar la tarea.

IV. Experimentación

IV-A. *Entorno de trabajo*

Los principales componentes de *software* utilizados fueron *Ubuntu 18.04*, *ROS* y *Cassandra*. La funcionalidad para su integración fue provista por el paquete *cassandra_ros*, como se mencionó en la Secc. III-C. Finalmente, para los componentes físicos, se dispuso de:

- una *Raspberry Pi 3 modelo B*[16] que ofició de robot generador de datos
- una computadora personal, en la cual se ejecuta *Cassandra 3.11*, para dar soporte como servidor de base de datos
- una computadora portátil, actuando como terminal independiente en la cual se postprocesan los datos

En la Fig. 3 se muestra una posible interacción entre estos componentes. Notar que en nuestro caso en particular existen dos salvedades: en la figura hay un robot consumidor de datos que no es mencionado en el entorno de trabajo (fue agregado por completitud del ejemplo, simplemente); y la terminal para procesamiento, en nuestro caso, solamente consume datos en forma de video.

[16]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

IV-B. *Configuraciones utilizadas*

Al ser entornos de desarrollo prácticamente homogéneos (la mayor diferencia fue la arquitectura del robot y los demás componentes), los comandos de instalación y las configuraciones para las componentes compartidas son iguales en todos los nodos. Se puede acceder a esta información de manera pública en el proyecto *tscf*[17] en GitHub, bajo el directorio `scripts/`. En ese mismo directorio se encuentran, además, el archivo de configuración utilizado para *Cassandra* (*cassandra.yaml*), y los archivos necesarios para automatizar el inicio de la red inalámbrica en el robot (bajo el directorio `scripts/raspberry_pi_3/`). Además, bajo el directorio `scripts/misc/` se dejan archivos misceláneos de instalación y configuración, que podrían ser de utilidad en futuros proyectos (como ser, pasos de instalación en una *BeagleBone Blue*[18]).

IV-C. *Pruebas realizadas*

Las primeras pruebas fueron realizadas en el servidor, mediante una cámara *USB* conectada directamente en uno de sus puertos. De esta manera se pudo probar que todos los componentes estuvieran funcionando correctamente, antes de probar en un entorno distribuído. En la siguiente etapa, se probó agregando la computadora portátil, reproduciendo remotamente el video previamente guardado en la base de datos (las pruebas fueron realizadas dentro de la misma red local). Posteriormente, se introdujo el robot, en el cual se utilizó la cámara *USB* para grabar el video. El caso de uso final fue el mencionado previamente: el robot graba un video y lo manda al servidor remoto, guardándolo en la base de datos, para luego ser reproducido en la computadora portátil. De esta manera, se cumple el ciclo de interacción remota, en el cual se le delega a la base de datos la gestión de los datos. Finalmente, para completar las pruebas, se utilizó comunicación entre los componentes vía Internet, demostrando que también es posible una comunicación remota eficiente.

Previo a decidir utilizar la *Raspberry Pi 3*, se intentó utilizar una *BeagleBone Blue* para las pruebas. Lamentablemente, no se tuvo éxito en enviar el video por la red (incluso vía la red local del servidor). Un diagnóstico superficial, mostró que se podría deber a la poca calidad de la conexión de red, mostrando la *BeagleBone Blue* tasas de transferencia y *ping* mucho menores a las de su contraparte. En el Cuad. I se muestran los valores promedio de transferencia para un archivo de `10Mb`, y *ping* para ráfagas de 10 paquetes de control. Se puede observar que la diferencia en ambos casos es de un orden de magnitud, siendo la *Raspberry Pi 3* la que mejores prestaciones ofrece. Si bien no hubo tiempo de analizar más detenidamente el motivo de la incapacidad de la *BeagleBone Blue* para funcionar correctamente, no se pudo detectar paquetes de datos entre la misma y el servidor en ninguna de las pruebas realizadas (utilizando *tcpdump*[19] para

[17]https://github.com/guriandoro/tscf
[18]https://beagleboard.org/blue
[19]https://www.tcpdump.org/

	Transferencia	***ping***
BeagleBone Blue	275 KB/s	3 ms
Raspberry Pi 3	7 MB/s	0.3 ms

Cuadro I: Valores promedio de transferencia y *ping* entre los computadores y el servidor central, vía red interna.

capturar los mismos), por lo que se decidió dejarla de lado en favor de la *Raspberry Pi 3*.

IV-D. Resultados obtenidos

Se pudo mostrar una aplicación real en la que una interacción fluída entre sistemas robóticos y sistemas de bases de datos es posible. Además de facilitar el manejo y la gestión de la información, se provee de funcionalidades adicionales que dan soporte a las tareas a ser llevadas a cabo por los robots.

V. Conclusiones y trabajos futuros

Se cree que las bases de datos tienen un lugar dentro de los sistemas robóticos, aportando funcionalidades como gestión y mantenimiento de datos, y facilitando el acceso a ellos. Además, pueden aportar en otras áreas, como ser la autenticación de clientes y la seguridad de los datos (encriptando no solamente la comunicación, sino que también los datos en disco).

También surgen nuevos desafíos, propios de integrar una nueva capa de tecnología al conjunto (*stack*) de soluciones. ¿Cómo se manejan la consistencia, integración y frescura de los datos? ¿Qué tipo de gestión de tráfico, o técnicas de prevención de saturación de la red deben ser utilizadas? ¿Qué otros modelos de bases de datos pueden utilizados, y cómo? ¿Qué tanto sobrecoste tiene mantener estos nuevos sistemas y sus interacciones? Estas preguntas quedan abiertas a trabajos futuros que puedan ser derivados de este análisis inicial.

Referencias

[1] P. Simoens, M. Dragone, and A. Saffiotti, "The Internet of Robotic Things: A review of concept, added value and applications," 2018. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/1729881418759424

[2] A. Dietrich, S. Mohammad, S. Zug, and J. Kaiser, "ROS meets Cassandra: Data Management in Smart Environments with NoSQL," 2014. [Online]. Available: https://www.semanticscholar.org/paper/ROS-meets-Cassandra-%3A-Data-Management-in-Smart-with-Dietrich-Mohammad/64f3b29939c0851563f18f93e3915a2cf12eee34

[3] A. Dietrich, S. Zug, S. Mohammad, and J. Kaiser, "Distributed Management and Representation of Data and Context in Robotic Applications," 2014. [Online]. Available: https://www.researchgate.net/publication/265998799_Distributed_Management_and_Representation_of_Data_and_Context_in_Robotic_Applications

[4] P. Mazzara, L. Steinfeld, A. Otero, F. Silveira, C. Saravia, and G. Fierro, "Redes de sensores inalámbricos aplicadas a la investigación y producción citrícola." [Online]. Available: https://iie.fing.edu.uy/investigacion/grupos/microele/papers/citrus10.pdf

[5] M. Ayyash, Y. Alsbou, and M. Anan, "Introduction to Mobile Ad-Hoc and Vehicular Networks," 2015. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-1-4939-2468-4_2

[6] R. Magán-Carrión, J. Camacho, P. García-Teodoro, E. F. Flushing, and G. A. DiCaro, "A Dynamical Relay node placement solution for MANETs," 2017. [Online]. Available: https://doi.org/10.1016/j.comcom.2017.10.012