Universidad Autónoma del Estado de Morelos

Facultad de Contaduría, Administración e Informática

Diseño de una Arquitectura Escalable Distribuida para Procesamiento de Datos de Geolocalización

T E S I S

que para obtener el título de:

Licenciado en Informática

presenta:

Carlos Alan Cervantes Salazar

Directores:

Dr. Pedro Moreno Bernal
M.T.I. Juan Manuel Hurtado Ramírez

cuernavaca, morelos mayo de 2022

Cuernavaca, Morelos a 14 de febrero del 2022
FCAeI/058/2022

C. CATEDRATICOS DE LA FCAeI
DE LA UAEM
P R E S E N T E

Me permito comunicarles que han sido designados como integrantes del jurado revisor de la **Tesis**, denominada: **"Diseño de una arquitectura escalable distribuida para procesamiento de datos de geolocalización",** con el cual pretende titularse el **C. Carlos Alan Cervantes Salazar** en el proceso de titulación de la Licenciatura en Informática, fungiendo como asesor el **Dr. Pedro Moreno Bernal,** director de Tesis.

El jurado quedo integrado de la siguiente manera:

1. LIC. ANA LINDA PINEDA MÉNDEZ

2. DR. JOSÉ ALBERTO HERNÁNDEZ AGUILAR

3. DR. GUSTAVO MEDINA ANGEL

4. DR. FEDERICO ALONSO PECINA

5. DRA. MIREYA FLORES PICHARDO

Ruego a ustedes tomar nota que, de acuerdo con el Reglamento de Titulación Profesional los miembros del jurado revisor tienen la obligación de formular la aprobación o rechazo del trabajo, dentro de los siguientes 30 días.

Atentamente

Por una humanidad culta
Una universidad de excelencia

MTRO. FELIPE DE JESÚS BONILLA SÁNCHEZ
Director de la Facultad de Contaduría, Administración e Informática

C.i.p. – Archivo.
GAOS

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

GERARDO ALEJANDRO OROZCO SIERRAS | Fecha:2022-02-14 16:06:23 | Firmante

m+CSJB1UgzzZbw/Qxo+5UK+iorUZxfNyCS20XpmjjlcGskcX9St5iMXA5minbZ5FSrkRcFGw25Meq9Zhed9iCqY2o5vCnQzVIvx0FUB6LBXjTfHorRboYc9mLz0hdOJZ68QL5M0niKvQ7NICGfeOGklHw6CbVtXnAYZ28eQIF5t54NBp9FZW8SgQdtJ/EroIKbXmy9L/vina/SqnassL33LO3O2QWO9fdHkJliRB9D2iB9jKDlS48PDOkr+TAZwmEkVA808Vt196GvkxCk2Q5VgY3QJHKytJ/PtnqTqTUFkpfqd/h2s9HxA0BjdR0GkMDwaAW24wT8AElFxQ1JMVGA==

ANA LINDA PINEDA MENDEZ | Fecha:2022-02-14 17:14:51 | Firmante

Es+H0YkuiGfWW2gFzBfV/DrMb3vEH3sb9SKBqxw0HUfOld4e0JOSKz9yUKlkpAAoDi12YaS7P2YuRvX0Xftt1GOvOmT9CkzETBEAiVR01UuXgkTmxJhjbENiyNFoirYkq92JJgOAD5qgf5vnu7Ac5dHwYt95F39uYdhX2HYHLzJwtld9TADPTXR2fCgowF7HCMB2cg3F+Rw6h2XHZg14B9wxwiib+cjswaM046oz5O3ejklEAwb5ZjYJzApoTlrODGSZCuP5gwbZ6ujz92e02E9w96DSH6jadcmLguIgQxEP2SigNpyLFM4DnPyJDUdVyU0t0vdY8wtPaFGAvY0r2w==

JOSE ALBERTO HERNANDEZ AGUILAR | Fecha:2022-02-14 19:34:34 | Firmante

B669u5nUJqUidQOW+Ybx7cg4ZRZ12NdE0f1HeSt5ZRs6RqsTUCi92hjlNkn0j7ePMN0yUdWKT6WRHqzOjpd33VxxeLE7Y+i6s2tQ7Bwbv5j8DmnUKHZMpSNJvZHGh8ucodOdGYtwXt1H1BsmRfoDpV3PGpRyh/B6GS58qSw2bWSL+UYcb95iuhPKvGsvDFvNsZ9OfHP/T3e7BHr/XGKY8FJvv29zcz5Ac7q4kJSz18SP4fYBm0FRu236v25iVloYd7XI4VIkPxT6FSiOUJgSvfoiNRdKQOwLUDBUilkPdJP1xvni0aCQyCYNKwQ+/R54+EfBxQoy2aqzwqNxnojyzw==

GUSTAVO MEDINA ANGEL | Fecha:2022-02-14 19:44:35 | Firmante

uYOtVw5Frjjmpog+cvZbSXypYW+2L1lgJEocsHn7zgfOsDtcUnM0HqJKltC8XIKsg157572h46cxs/WQRtlskDKRvAtk8hShCvm0e8nP5+qOM61cpYzTL/kozwsTpNjd8TtgNaRYJLKiSZKSRcqYIQD2aN9zaWOXIE5/b/xXH3NgiSfXsLmYMpkqL9vkaIZGIdtTFQahFIZMI4++MBlwIPvvLp5f7812ao6dp1Roh8ihFsxSQa3WMhQ6WVxAdu64ypzfefKh4cCrXv6GLjsf/j7yhqgREMFpoc9IPQPlKGJwSohXjIj2T5EAR0DRvgnnKYocPfSBkOPMOWJpSh5W0w==

FELIPE DE JESUS BONILLA SANCHEZ | Fecha:2022-02-14 20:10:22 | Firmante

KJMF98BwS3dq5jf/0RbWeKvhp0fSHlRFIpleuJbZoW2BHpPXf92Y6053nz0AQzvXgmggcMnJ+Y38YyY53XW9S8QzzQxcnO8asD1Z7RLZZNJQ6A1EW0mc9J/Y0zTaKgCDD9LCuIj1j+3iedDmpmRjYQMGr9CfkSHgALDI3qGabQx2L3JGAs8ScmmOpd0piwGpdKzudbsSsBR9hGEbCWm+mdLBI9FER5L915FtuWUNXQi0P1ttZOZMQKCQkPz5M+nxnbybtvueKbxtgfmc11Q5LnBZMoG5oU0lep6BSUtKEyOgV4y7aAh9zQkpt9elT94cb05VxhZF4fVzCd6wifyRrg==

MIREYA FLORES PICHARDO | Fecha:2022-02-15 11:46:58 | Firmante

0Njs6oIlxUdObOqKvrDtQxPSdh8lW3FNZN5k8L4RiZttencbJ5leAkDTD/gofhqRDrVp8A2XDYyKwHWBgESELND+f6UsGtzCcCGwV7v+IlCwG7mmkL6CRwiWcD7SmOe5LqR1Mfj371UBMJCLxP42L2Ca1Z48+lFPbmkVi/Unq+YJhD3ypKV3DmnBD1gmTMg2YFdsW12U4Qv+ZJKgU+w3tdrxCR3qiNJuWZ3MdkCNU5bebP0cu67L7zP3HACSyVGFWGTDPxn7210SISgHAEaU2YPf4Def2Rtqab9kSPKRn7MLk4AbuBCtv8U7N137yfYk1r1Vj+QPcJ87QScOpaP6Xg==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

jO5YXZTUQ

https://efirma.uaem.mx/noRepudio/AVR8MTdwNBQWBURggmvyQClhYSvFJ1yn

Cuernavaca, Mor., a 16 de Marzo del 2022

MTRO. FELIPE DE JESÚS BONILLA SÁNCHEZ
DIRECTOR DE LA FCAeI
DE LA U.A.E.M.
PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: "**Diseño de una arquitectura escalable distribuida para procesamiento de datos de geolocalización"**, que presenta el C. **Carlos Alan Cervantes Salazar**, Pasante de la carrera de la **licenciatura en informática**, a mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarles mi **voto aprobatorio**, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

_____

Dr. José Alberto Hernández Aguilar

C.c.p. **Mtro. Miguel Ángel García Garnica.** Secretario de Extensión de la FCAeI.
C.c.p. **Mtro. Ángel Estrada Arteaga.** Secretario Académico de la FCAeI.
C.c.p. **Lic. Dora Irma Barrios Alvarado.** Jefa de la carrera de la licenciatura en informática.

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

JOSE ALBERTO HERNANDEZ AGUILAR | Fecha:2022-03-16 23:05:42 | Firmante

zqRCs/CMqYqxT6P/slX+YrozaLif38j9RIPFYoaPjVtnzm3ohcxcZ4aKinzCAlcG6E5NA5hXUn9PVd2nfC1qZl/pjEVXvJll4b78ChguL489psliiLKFxypZU1ei/P+dcUIapUtwyDxiBtNMVD28kIYvJ7wTwrdi9hHNvbVZ9jKaxpjDp4+Le1Ft167v1ky+9Odj0rvY7i74YgydF8fC4ie8t76OnlSHr7ECV+YfJrKYTsxBJpqSfYvOB97DDFYHWAc0SXiMf2OYah/ngB+/yi/R3L/rgP9jldrdJ4sQdGt1VjTR0PMvOixaUPNVnSPKcw7uj6F60fCWEhOvjsMq1A==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

ewJZNY2ET

https://efirma.uaem.mx/noRepudio/lyJ3fpQ95PdMI2MnVpyL56dZzam7kGw3

UNIVERSIDAD AUTONOMA DEL ESTADO DE MORELOS
FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos, a 16 de marzo del 2022

MTRO. FELIPE DE JESÚS BONILLA SÁNCHEZ
DIRECTOR DE LA FCAeI
DE LA U.A.E.M.
PRESENTE

Por este conducto, me permito informar a usted que la tesis denominada: "**Diseño de una arquitectura escalable distribuida para procesamiento de datos de geolocalización"**, que presenta el C. **Carlos Alan Cervantes Salazar**, Pasante de la carrera de la **licenciatura en informática** de la Facultad de Contaduría, Administración e Informática (FCAeI), a mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarles mi **voto aprobatorio**, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

_____

Dr. Federico Alonso Pecina
PITC de la FCAeI

C.c.p. **Mtro. Miguel Ángel García Garnica.** Secretario de Extensión de la FCAeI.
C.c.p. **Mtro. Ángel Estrada Arteaga.** Secretario Académico de la FCAeI.
C.c.p. **Lic. Dora Irma Barrios Alvarado.** Jefa de la carrera de la licenciatura en informática.

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

FEDERICO ALONSO PECINA | Fecha:2022-03-16 13:59:58 | Firmante

vj+nZgIJ6F4N+0e0vLCFZ2W21zo1Y7FN3IVZ2iuDAOMHIHCFvZlSAJuLkTei94p7rQ0r/tt+BaU1gG0X0h9tvv6BRMcSEfvS8t9jKj/BXnYQquMOppTF0/m1GBW3wJApbS2l8V8QlHksBDAvJdwZFoKK8skPQzFqjA+4wX4AAvaafWVvgzzZr9mAALTU/hOhCwO4jqI23/8mTDowprFvigwIbkDUR7z8qQeP4DvESVhQr2Es3Inl0w06KmU+E5Vs3Z7+KYfaXb6yq70UAxfmSgLkAKeZPImOtFL8XAjXKZcugnNTnB2tmYuqWUPyqQJRFs+7JbJxQuU21NWSN0ln2w==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

syj8W9hoT

https://efirma.uaem.mx/noRepudio/OAq7BNf9YIV4T7QF4wiPTgWY0ltwEzk3

Cuernavaca, Mor., a 01 de marzo del 2022

MTRO. FELIPE DE JESÚS BONILLA SÁNCHEZ
DIRECTOR DE LA FCAeI
DE LA U.A.E.M.
PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: "**Diseño de una arquitectura escalable distribuida para procesamiento de datos de geolocalización"**, que presenta el C. **Carlos Alan Cervantes Salazar**, Pasante de la carrera de la **licenciatura en informática**, a mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarles mi **voto aprobatorio**, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

DR. GUSTAVO MEDINA ÁNGEL

C.c.p. **Mtra. Miguel Ángel García Garnica.** Secretario de Extensión de la FCAeI.
C.c.p. **Mtro. Ángel Estrada Arteaga.** Secretario Académico de la FCAeI.
C.c.p. **Lic. Dora Irma Barrios Alvarado.** Jefa de la carrera de la licenciatura en informática.

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

GUSTAVO MEDINA ANGEL | Fecha:2022-03-01 00:43:11 | Firmante

HtuGBydmfvcwwhCQANgmjB00s+53l4mutubodRKGSyWx1mjNyJ69XrX35xFC7gIdJYVvkhWsZRyYu+lMawEha/Rg57WcOovrrt33qp5tK+W8Mi2gwwZdDnQJlKfrtIent0/k4TKtfi2K
BrytEsp9BdRMgtS95O+6kfqVQk4ooWCmzqeZX2KHToMaseiiDpoVizskyLpQyw1WT5/hJKeZ9mtEfDMun6/Ceje9AxdtJjBXOi5R3Gh5kiLT3KbEiHaaptav9xe8j26sFPTA22rvus16X
RP3K9yUMq5inBc889pM0D27L1zshY9a3GD3UBF2aC3qqh6zcUNiD6lWndlxRA==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

Sf7vgmdo9

https://efirma.uaem.mx/noRepudio/mFG7uXMofZ8JpZZtjS39xMLhFrXIxRrI

Cuernavaca, Mor., a 16 de marzo del 2022

MTRO. FELIPE DE JESÚS BONILLA SÁNCHEZ
DIRECTOR DE LA FCAeI
DE LA U.A.E.M.
PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: "**Diseño de una arquitectura escalable distribuida para procesamiento de datos de geolocalización"**, que presenta el C. **Carlos Alan Cervantes Salazar**, Pasante de la carrera de la **licenciatura en informática**, a mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarle mi **voto aprobatorio**, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

_____
Dra. Mireya Flores Pichardo

C.c.p. **Mtra. Miguel Ángel García Garnica.** Secretario de Extensión de la FCAeI.
C.c.p. **Mtro. Ángel Estrada Arteaga.** Secretario Académico de la FCAeI.
C.c.p. **Lic. Dora Irma Barrios Alvarado.** Jefa de carrera de la Licenciatura en Informática.

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

MIREYA FLORES PICHARDO | Fecha:2022-03-16 12:39:51 | Firmante

1y6gBfPgumBD4vC1LD1zd6JLNXKAhw5kty8p7bMcQo7vW8TYlLAJ029VjV3kIQVPzSEIAGJU3wqhmJD4au3o9l8IsE78fJocs9GIqpQct7m3orutgsyuvXnUmcHR3bd7+2aYrf89W/BmCQm1mYdOO4+sYLaJta4qWWt4FnYmLO/BC4uXDfxft4AzPtdcdkPxCB9dqRuzV+B3fvPOEiEPNbPRL6UdxtfZZwQi6V7mR10X8QYtOzstXUODCRIlqtom03ZQpiNCyfIhFWfwyeyUX81QDF6jsEJCQDKtCeNFQAWpAmozmi600SS40mcOfjnZgpPfAaCcwTnanho0TqYZAg==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

rhEPUfVnK

https://efirma.uaem.mx/noRepudio/z13bB9Yq2xOZFxqB8hCRr4ragrJfDFvH

Cuernavaca, Mor., a ____16_ de _____marzo_____ del ___2022_____

MTRO. FELIPE DE JESÚS BONILLA SÁNCHEZ
DIRECTOR DE LA FCAeI
DE LA U.A.E.M.
PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: "**Diseño de una arquitectura escalable distribuida para procesamiento de datos de geolocalización"**, que presenta el C. **Carlos Alan Cervantes Salazar**, Pasante de la carrera de la **licenciatura en informática**, a mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarles mi **voto aprobatorio**, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

M en E. Ana Linda Pineda Mendez

_____

(Nombre y firma)

C.c.p. **Mtro. Miguel Ángel García Garnica.** Secretario de Extensión de la FCAeI.
C.c.p. **Mtro. Ángel Estrada Arteaga.** Secretario Académico de la FCAeI.
C.c.p. **Lic. Dora Irma Barrios Alvarado.** Jefa de la carrera de la licenciatura en informática.

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

ANA LINDA PINEDA MENDEZ | Fecha:2022-03-16 20:48:46 | Firmante

Tr5EwrXjYimiivzGJ/MsK7qh0hJoyyfEBHp4eDpPL/r55C/fDwgEShvo9ynO39RTytYY23oJO7eOulyNXSVVaECs6XX6RaC2gOG6HGYH1Hzf4m3bCTQ6bTvHbPToE95TPWk7BJiyk5lDymUAqXcSH08DbaGeVzcESPcSIwxDgculB3MUhpyB5kLN6i1qjthRflHYUysRxf0SJdTTGbB2b6/N8Ycaw2Lg/Tzr2Y7vBw+tCw1ILNsTvUj324sDD/ekwiWyWgPa+HrzuTI3UJFUpcU6/6+znP6q2iW3QQoNZTFAjSGqVY3ssdVuyptD1SuMzFdQxkTVOXZswgy+/YRwvQ==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

LbWVQGT4D

https://efirma.uaem.mx/noRepudio/Ht2JyZcsfAOrnWmu1gS5rvW32oJjBjvH

Este trabajo tiene una dedicatoria especial para mi mamá, Angelina Salazar Villegas, por formar esta hermosa familia y siempre luchar por nosotros. Le estoy eternamente agradecido por regalarnos su amor día a día, cuidar de nosotros y ser la mejor mamá del mundo. Eres es una inspiración y ejemplo a seguir para todos. Te amo mucho, ma.

Por supuesto, también va dedicado con mucho cariño a mi abuelita, Angelina Villegas Tapia, por su amor, su apoyo incondicional y por siempre estar presente. Gracias por confiar en mi. Te amo mucho, abue.

Agradecimientos

A toda mi familia, en especial a mi mamá, a mi abuelita y a mi hermana. Sin ellas, nada de esto sería posible.

A mi docente y asesor, por su dedicación, orientación, por los conocimientos brindados y confiar en mis capacidades para llevar a cabo este proyecto.

A mi co-asesor, por las facilidades y atenciones otorgadas durante el desarrollo de este trabajo.

A mi casa de estudios, la Universidad Autónoma del Estado de Morelos, por permitirme vivir la mejor etapa de mi vida estudiantil.

A los docentes de la Facultad de Contaduría, Administración e Informática, por su compromiso para formar licenciados en Informática preparados y comprometidos con la sociedad.

A mis amigos y compañeros de grupo, por las experiencias vividas a lo largo de estos años.

A mi novia, por estar conmigo, apoyarme y alentarme a seguir adelante.

Resumen

Actualmente, la información es la base del avance de las Tecnologías de Información y Comunicación (TIC). Cada día, grandes volúmenes de información son generados por millones de personas en todo el mundo que envían y reciben datos a través de los distintos servicios y aplicaciones confiables disponibles en internet, así como los datos producidos por el entorno de las organizaciones, conformando las 3V's: Volumen, Variedad, Veracidad; las cuales conforman las bases de lo que se conoce como Big Data. Aunado a ello, el Internet de las Cosas (IoT, por sus siglas en inglés) genera datos que se transfieren entre sí a través de miles de millones de objetos o cosas que están comunicadas a través de internet.

La información debe estar disponible en todo momento, es decir, se debe poder acceder a los datos, conocer dónde están almacenados y saber quién los posee dada su importancia. La disponibilidad de los datos permite realizar acciones de suma importancia para las empresas u organizaciones, por lo que se hace necesario diseñar arquitecturas distribuidas para procesamiento y análisis en tiempo-real de grandes cantidades de datos para analizar y clasificar datos.

Este trabajo introduce el diseño e implementación de una arquitectura distribuida escalable utilizando software libre para el procesamiento de seguimiento de geolocalización en tiempo real a partir de la ubicación GPS (Global Positioning System) de un dispositivo IoT. El diseño recolecta información a través del protocolo MQTT (Message Queue Telemetry Transport) para almacenar la información en Apache Kafka. Posteriormente, la información es procesada mediante Apache Storm para garantizar tolerancia a fallas, escalabilidad horizontal y baja latencia, para finalmente almacenar la información en una base de datos para futuro análisis. La evaluación experimental se realizó considerando 10357 trayectorias de taxis usando diferentes tamaños de instancias del problema. Los resultados muestran que el diseño de la plataforma propuesta es capaz de procesar en tiempo razonable un significativo numero de estampas de tiempo GPS de las instancias probadas.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN

1.1. Introducción

Actualmente, la revolución de Big Data está avanzando y el Internet de las Cosas impulsa el avance como parte esencial de una era trasformadora de cambios tecnológicos bajo el paradigma de ciudades inteligentes [1]. Como parte de los cambios tecnológicos, la movilidad urbana debe proveer servicios de movilidad inteligente que anticipe cambios futuros y participe en la toma de decisiones. La movilidad en ciudades grandes es compleja por el tráfico, por los distintos modos de transporte y por los múltiples orígenes y destinos de los desplazamientos de movilidad [2]. En años recientes, la contaminación y el tráfico se han convertido en los principales problemas para las ciudades más importantes. La congestión del tráfico es un problema severo debido a que genera pérdidas importantes en la economía e impacta en la calidad de vida de los ciudadanos. Por este motivo, es necesario construir sistemas de gestión de transporte inteligente mediante seguimiento de movilidad urbana para mejorar la eficiencia y seguridad del transporte urbano. Además, los sistemas de transporte deben garantizar la seguridad y eficiencia de los viajes, evitar impactos negativos en la economía, reducir la contaminación ambiental, rastrear el movimiento e impactar la calidad de vida de las personas, convergiendo en Sistemas de Transporte Inteligente(STI). Los STI son sistemas de movilidad que utilizan avances tecnológicos, métodos y aplicaciones del campo de las Tecnologías de la Información y la Comunicación (TIC), con alta tecnología y movilidad de datos. Las TIC recopilan una amplia gama de datos de senso-

res, infraestructura y geolocalización de dispositivos de movilidad para operar y administrar científicamente [3]. Además, las TIC transfieren de manera eficiente los datos recopilados a un centro de datos para compartir e intercambiar información entre dispositivos conectados [4]. Los datos de geolocalización se generan continuamente a partir de dispositivos IoT, y estos flujos de datos son de alta granularidad temporal [5]. Todos los días, la cantidad de datos de movilidad crece a un ritmo exponencial.

El big data integra altas tecnologías y grandes cantidades de datos de las TIC para desarrollar nuevos sistemas y aplicaciones digitales en tiempo real para analizar, simular y procesar datos de movilidad. Las técnicas de big data requieren análisis efectivos, tecnologías de almacenamiento y herramientas informáticas de procesamiento distribuido para explotar grandes cantidades de datos en la nube. El análisis y procesamiento de grandes cantidades de datos ha llevado a las organizaciones a desarrollar plataformas de soluciones distribuidas para maximizar los beneficios del valor de los grandes volúmenes de datos. El big data debe ser capaz de procesar datos de Terabytes a Zettabytes a una velocidad alta y casi en tiempo real para explorar y explotar el poder de big data [6]. En el análisis en tiempo real, los dispositivos móviles envían continuamente grandes volúmenes de datos y los servicios en línea los reciben en la nube para su procesamiento, análisis y almacenamiento. Las herramientas de big data utilizan modelos de métodos iterativos, de flujo y por lotes para procesar datos a gran escala. El modelo seleccionado depende del tipo de problema a resolver.

Hoy en día, los datos de localización de un vehículo en movimiento son recopilados para monitorear en tiempo real su trayectoria desde su origen a su destino. El problema de ubicar vehículos en tiempo real es crucial. En ciudades con alta densidad de población, el tráfico vehicular y la contaminación del medio ambiente, principalmente del aire y agua impactan negativamente en la economía y la calidad de vida de sus habitantes. Por esta razón, contar con STI que reduzcan la saturación del flujo vehicular, garanticen la seguridad e incrementen la eficiencia de los viajes es una medida imprescindible para ciudades con actividades de movilidad altas [5].

1.2. Antecedentes

La investigación en ciudades inteligentes utiliza tecnologías de big data e IoT para apoyar los STI en la planificación de la movilidad urbana. Muchas aplicaciones STI utilizan herramientas de análisis de big data para el procesamiento de datos en tiempo real. Los trabajos actuales proponen diferentes arquitecturas y plataformas para recopilar, procesar, almacenar y analizar datos de movilidad urbana en el contexto de las ciudades inteligentes. A continuación se presenta una breve reseña de trabajos relacionados.

Hongyu Wang et al. [7] proponen un algoritmo para buscar coincidencias de coordenadas de geolocalización en tiempo real. Para esto, hacen uso de datos masivos de mapas del Sistema de Posicionamiento Global (Global Positioning System, GPS) que garanticen una alta precisión y eficiencia de coincidencia. El algoritmo de coincidencia topológica modificado aplica el filtro de Kalman para corregir y mejorar la precisión de los vehículos en el mapa. El algoritmo se paraleliza para analizar la capacidad del manejo de flujo de datos en tiempo real y la tolerancia a fallas entre Apache Storm y Apache Spark. El método de cuadrícula o discretización de malla se aplica al algoritmo paralelo para mejorar la eficiencia del cálculo. El trabajo utiliza Apache Spark como framework de cálculo para la correspondencia de mapas en tiempo real. El algoritmo paralelo se ejecutó en un clúster de 15 computadoras con datos de coincidencia de mapas en tiempo real usando datos GPS producidos por 800 mil vehículos de manera efectiva. Los resultados experimentales muestran un aumento del 10 % en la precisión y del 25 % en la eficiencia de la coincidencia en las coordenadas geográficas.

Ayae Ichinose et al. [8] proponen un framework para el análisis de vídeo recopilado de múltiples cámaras usando Apache Kafka y Apache Spark. El sistema de procesamiento de flujo de datos en tiempo real de alto rendimiento incluye una cola de mensajería en Kafka para la ingestión de datos y un motor de procesamiento basado en Spark. El trabajo analizó el rendimiento de transferencia de datos en Kafka. Igualmente, se evaluó la configuración eficiente de los parámetros de ejecución en un clúster. La configuración del framework propuesto midió el rendimiento del análisis de datos. Los resultados experimentales mostraron

que el rendimiento general varía según el número de nodos de brokers que almacenan datos, el número de particiones de tópicos de datos y el número de nodos que realizan el procesamiento de análisis. Además, se confirmó que se debe considerar el número de núcleos para una configuración eficiente del clúster, y que el ancho de banda de la red entre los nodos se convierte en un cuello de botella a medida que aumenta la cantidad de datos y el número de componentes.

Marius Laska et al. [9] proponen una arquitectura para el manejo de flujo de datos espacio-temporales en tiempo real en dispositivos de Internet de las Cosas. Se utiliza Apache Storm como motor de procesamiento de flujo, GeoMQTT como protocolo de mensajería y Apache Kafka como fuente de entrada entre el clúster de brokers GeoMQTT y Storm. De esta manera, se aborda la integración de datos y el procesamiento de flujo escalable, distribuido y tolerante a fallas, implementando un algoritmo de coincidencia de mapas en línea. La arquitectura propuesta, conecta dichas herramientas para construir un canal de comunicación de datos. El protocolo GeoMQTT crea un clúster de brokers para procesar el flujo de datos espacio-temporales. Los mensajes publicados se reenvían a un clúster de Kafka para que Storm los consuma más adelante. El puente GeoMQTT-Kafka hace una depuración de mensajes específicos para el motor de procesamiento de flujo y ofrece un balanceo de carga en el punto de entrada de flujo de datos. El framework de Storm se utiliza para implementar adaptaciones distribuidas de algoritmos de correspondencia de mapas en línea conocidos, que consumen mensajes del broker de Kafka. Los resultados de los algoritmos de coincidencia de mapas en línea se publican nuevamente a través de un grupo de brokers de GeoMQTT de tamaño variable y son consumidos por cualquiera que se suscriba al tópico de salida. Los resultados experimentales presentaron latencias estables en el umbral de los milisegundos al no exceder un índice de publicación simulado de 100 puntos de trayectoria por segundo.

Zdravko Galic et al. [10] proponen un framework paralelo para el seguimiento y análisis en tiempo real de objetos móviles. Esto se realiza a través del procesamiento de flujos de datos y un modelo de consultas espacio-tiempo. La plataforma subyacente utiliza Apache Flink para construir el framework mediante diferentes niveles de abstracción. La capa de

APIs genera JobGraphs (Grafo Acíclico Dirigido, DAG) para sus programas a través de procesos de compilación separados. Se aplica DataSet para determinar y optimizar el plan del programa y, DataStream para construir un flujo de datos continuo, paralelo e inmutable. El núcleo de Flink procesa y canaliza los datos de movilidad según la hora del evento. La capa de ejecución recibe el JobGraph que se ejecuta de acuerdo a una variedad de opciones de implementación disponibles como local, remoto u otro negociador de recursos. El modelo discreto de consultas presenta un objeto espacio-temporal como un conjunto de unidades espacio-temporales cuyos intervalos de tiempo son disjuntos y, si son adyacentes, sus valores son diferentes. Una clase Case en Scala es definida para agregar una fuente de datos o para mapear una lectura de datos desde un socket a un flujo de datos derivado. El análisis experimental evalúa y valida un enfoque de formulación de consultas y simula un escenario del mundo real usando un conector Apache Kafka para acceder a los datos de movilidad de GeoLife. El tiempo de entorno de ejecución del flujo se establece en un tiempo de evento. Esto, para relacionar las marcas de tiempo con los eventos. El framework paralelo demostró ser eficiente al procesar muchas operaciones para grandes volúmenes de datos móviles y el soporte nativo para el procesamiento de consultas espacio-temporales en tiempo real.

Nesmachnow et al. [11] describió una plataforma para el análisis de Big Data utilizando un enfoque Map-Reduce en Hadoop en el contexto de las ciudades inteligentes. Los datos de la instancia del problema incluyeron diferentes intervalos de tiempo y ubicaciones GPS de autobuses del sistema de transporte de Montevideo, Uruguay. Los tamaños de archivo de los conjuntos de datos evaluados son de 10 GB, 20 GB, 30 GB y 60 GB. El análisis experimental se centró en evaluar la eficiencia computacional del modelo Map-Reduce paralelo/distribuido y la corrección del sistema en varios escenarios utilizando datos GPS precisos recopilados en 2015 en Montevideo. La evaluación experimental se realizó sobre la infraestructura en la nube del Clúster FING, equipo de computación de alto rendimiento de la Universidad de la República, Uruguay [12]. Los resultados experimentales indicaron que el enfoque de la solución escala adecuadamente cuando se procesa un gran volumen de datos con una aceleración de 22.16 veces y una eficiencia computacional de 0.92 utilizando

24 recursos informáticos en comparación con su versión secuencial.

Massobrio y Nesmachnow [13] propusieron un estudio de análisis de datos urbanos para el sistema de transporte público de Montevideo en STI y ciudades inteligentes. El estudio analizó el conjunto de datos de ubicación de autobuses por GPS y la venta de boletos con tarjeta inteligente. Se obtuvieron varias perspectivas del análisis de datos, como el número de pasajeros que viajaban con la misma tarjeta inteligente, la frecuencia de uso de las tarjetas inteligentes y el número de transferencias de autobús. Asimismo, el análisis de venta de boletos reveló tres horas pico durante los días hábiles. Además, el trabajo propuso una metodología para construir matrices origen-destino por cambio de viaje para estimar los destinos. El algoritmo implementado estimó correctamente el destino del 81.62 % de los viajes de movilidad urbana del conjunto de datos estudiado de 2016 con un coeficiente de correlación Spearman de 0,895.

Los trabajos relacionados permiten identificar varias propuestas que utilizan plataformas de big data para el procesamiento y análisis de aplicaciones de STI en el contexto de las ciudades inteligentes.

1.3. Justificación

De acuerdo a la literatura, existen aplicaciones que se ejecutan en ecosistemas complejos que presentan problemas de organización frente a la arquitectura. Las dificultades se presentan en conectar sistemas que generan información con aquellos que la recopilan. Algunos inconvenientes son el acceso y el procesamiento de los datos, la comunicación entre protocolos de transporte, la gestión de grandes volúmenes de cargas por peticiones de conexión, entre otras. Una solución ante estos obstáculos es el uso de arquitecturas escalables distribuidas para el manejo de flujos de datos en tiempo real.

El presente trabajo se enfoca en el diseño de una arquitectura escalable distribuida para procesamiento de grandes cantidades de datos de geolocalización para ubicación GPS de vehículos de transporte, en la búsqueda de una alternativa que genere un impacto positivo en el sector de transporte en México.

1.4. Objetivos

Objetivo general. Diseñar e implementar una arquitectura distribuida escalable para procesamiento en tiempo real de grandes cantidades de datos de geolocalización.

Objetivos específicos

- Realizar un estudio de arquitecturas distribuidas para procesamiento de datos en tiempo real que permita gestionar grandes volúmenes de datos.
- Diseñar la arquitectura del sistema distribuido que sea escalable para procesamiento de datos de geolocalización en tiempo real.
- Analizar la eficiencia de procesamiento de datos en tiempo real con datos generados de manera aleatoria.
- Analizar y clasificar datos en tiempo real de la localización GPS de vehículos.

Metas

- Obtener el diseño de una arquitectura escalable del sistema distribuido para el procesamiento de datos de geolocalización.
- Obtener un análisis de eficiencia de procesamiento de datos en tiempo real.

1.5. Contribución

La contribución de este trabajo es el diseño de una arquitectura escalable distribuida para procesamiento de grandes cantidades de datos de geolocalización basada en Apache Kafka, Apache Storm y el protocolo MQTT sobre la nube. Asimismo, la implementación de persistencia de datos mediante el manejador de bases de datos MySQL.

1.6. Estructura de la tesis

El documento se estructura de la siguiente manera. El capítulo 2 expone el problema de transporte y geografía y la importancia del procesamiento en tiempo real de datos de geo-

localización. El capítulo 3 presenta las Tecnologías de la Información y Comunicación (TIC) usadas en este trabajo, así como los protocolos de mensajería para llevar a cabo el diseño de la arquitectura escalable y distribuida. El capítulo 4 describe el diseño e integración de la arquitectura escalable del sistema distribuido. El capítulo 5 describe el análisis experimental y la discusión de los resultados de rendimiento de la arquitectura propuesta. Finalmente, el capítulo 6 explica las conclusiones del diseño de la arquitectura escalable distribuida para el procesamiento de grandes cantidades de datos de geolocalización y propone las principales líneas de trabajo futuro.

Capítulo 2

DEFINICIÓN DEL PROBLEMA

Esta sección describe la importancia de los datos de geolocalización y cómo su procesamiento en tiempo real es esencial en el los Sistemas de Transporte Inteligente.

2.1. Transporte y geografía

En los últimos años, las actividades de movilidad urbana han crecido masivamente. Esto, ha derivado en problemas de congestión vehicular, contaminación del aire e incremento de accidentes de carretera [3]. El uso de las TIC aplicadas a los sistemas de transporte buscan reducir el impacto negativo que conllevan estas cuestiones a los habitantes de las ciudades con mayor densidad. Los STI surgen con la implementación de nuevas tecnologías y técnicas de movilidad efectivas. Los STI permiten monitorear la ubicación, velocidad y recorrido de un vehículo en tiempo real para brindar servicios que mejoren la fluidez del tráfico, garanticen la seguridad y la eficiencia de los viajes. La seguridad es un aspecto crítico en el diseño de los STI. Las aplicaciones deben realizar todas sus tareas en tiempo real. Los procesos exigen un tiempo límite específico para su culminación. Un fallo del sistema puede llevar a producir pérdidas humanas, económicas y ambientales [3].

2.2. Geolocalización de vehículos de transporte

Los servicios de transporte, como los taxis y las empresas de entrega a domicilio, requieren un seguimiento de la ubicación de sus activos para garantizar que sus vehículos

se aprovechen de la mejor manera. Por lo tanto, el monitoreo de los vehículos es crucial para localizar cualquier activo en caso de robo o si necesita modificar la ruta original. La interacción geoespacial es un elemento fundamental para el transporte urbano. Los datos geoespaciales en tiempo real ayudan a capturar los movimientos de los vehículos para proporcionar su información de geolocalización sobre la infraestructura de transporte [5]. Por ello, un enfoque de STI que use información de geolocalización se vuelve fundamental para un transporte urbano eficiente y seguro. Los STI deben proporcionar información confiable a los conductores sobre su entorno de conducción. La información de los vehículos y usuarios de la carretera ayuda a mejorar las condiciones de conducción para una movilidad urbana segura, eficiente, cómoda y más limpia. Se crea información confiable en todo momento, lo que aumenta la cantidad de datos sobre los vehículos, la ubicación, la infraestructura, las condiciones de la carretera y el operador [14]. Los datos recolectados permiten generar rutas optimizadas para una movilidad segura considerando la capacidad de la infraestructura de la red vial.

Esta información ayuda a atender incidentes y peligros mediante una mejor respuesta de movilidad. Además, los STI proporcionan mejores servicios para los diferentes modos de transporte, disminuyendo cualquier impacto económico negativo. La interoperabilidad de los STI requiere grandes cantidades de datos de movilidad, en particular, ubicaciones geográficas sobre la movilidad de los vehículos. Esto, permite garantizar la funcionalidad de los STI para la toma de decisiones públicas y privadas. Algunos de los beneficios de rastrear un vehículo son el control del programa de mantenimiento por kilómetros, la ubicación rápida del vehículo, la recopilación de datos para valorar el costo operativo futuro, la mejora del tiempo de actividad y la utilización del vehículo [2].

2.3. Procesamiento de datos de geolocalización en tiempo real

Las TIC, las tecnologías de sensores y los dispositivos digitales utilizan redes de sensores inalámbricos para recopilar datos geoespaciales en tiempo real automáticamente [15]. Las redes de sensores inalámbricos producen grandes cantidades de datos de geoloca-

lización en tiempo real. Además, el monitoreo de tráfico en tiempo real utiliza sensores montados en faros, debajo del pavimento, en vehículos (geolocalización por GPS), puentes, cabinas de peaje, postes de semáforos, entre otros, para recopilar ubicación, velocidad, direcciones de movimiento y datos meteorológicos para estimar las condiciones de flujo de tráfico en un contexto de movilidad urbana. Los datos de movilidad urbana en el contexto de las ciudades inteligentes brindan la creación de nuevas reglas, regulaciones y métodos para controlar el crecimiento de la densidad del tráfico vehicular. Además, los datos de movilidad urbana permiten la expansión y construcción de nuevas rutas, caminos e infraestructura de movilidad para resolver los problemas existentes que impactan significativamente en la economía local.

Los STI integran tecnologías de big data, técnicas de inteligencia computacional e ingeniería de sistemas de transporte para mejorar los servicios de movilidad del transporte. Los STI deben recopilar grandes volúmenes de datos en tiempo real mediante dispositivos IoT y redes de sensores móviles en vehículos e infraestructura de redes viales [11]. Además, las herramientas de big data para STI requieren datos de análisis adecuados, técnicas de almacenamiento y herramientas de procesamiento paralelo/distribuido para un procesamiento de datos eficiente y preciso (en tiempo real o fuera de línea). Estas tecnologías proporcionan herramientas para abordar los problemas que deben abordarse en el contexto de las tecnologías de almacenamiento, el procesamiento/análisis por lotes o el procesamiento/análisis en tiempo real [16]. Los frameworks específicos para el análisis de datos en tiempo real en la escala de Teradata y el procesamiento paralelo/distribuido en big data son Apache Storm [17] y Apache Kafka [18]. Ambos frameworks se caracterizan por la tolerancia a fallas y la escalabilidad para el procesamiento de datos de transmisión en tiempo real. Además, estos frameworks admiten diferentes etapas de transmisión, como la recopilación, el transporte y el proceso. Otros frameworks relacionados con capacidades similares con un enfoque de desarrollo funcional son Apache Spark y Apache Hive. El procesamiento de datos es el objetivo principal para calcular métricas relevantes para servicios de transporte eficientes.

Este trabajo se centra en plataformas distribuidas de procesamiento de big data utilizan-

do herramientas de código abierto para el seguimiento de geolocalización en tiempo real. El objetivo es rastrear vehículos basados en movilidad urbana para STI en el paradigma de ciudades inteligentes.

Capítulo 3

TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

Esta sección describe la plataforma de big data de código abierto propuesta para el seguimiento de geolocalización en tiempo real. Además, se describen los frameworks, protocolos, diseño y arquitectura distribuida.

3.1. Internet de las Cosas

El Internet de las Cosas o IoT puede definirse como una red de dispositivos conectados a Internet que generan, intercambian y procesan información en tiempo real [19]. Algunos ejemplos de dispositivos de IoT son teléfonos y relojes inteligentes, distintos tipos de sensores, antenas con Sistema de Posicionamiento Global (GPS, por sus siglas en inglés), entre otros. A medida que las TIC han ido avanzando en los diversos sectores productivos de la sociedad, esta serie de cosas u objetos interconectados requieren de arquitecturas que les ayuden a impulsar todas sus capacidades.

3.2. Protocolo MQTT

El Transporte de Telemetría de Cola de Mensajes (Message Queuing Telemetry Transport, MQTT) es un protocolo de mensajería M2M (Machine-to-Machine) basado en TCP/IP (Transmission Control Protocol/Internet Protocol) [20]. MQTT trabaja con el modelo Publi-

cación/Suscripción (Pub/Sub) a través de un servidor o broker. Los clientes (dispositivos de IoT) inician una conexión TCP/IP con el broker MQTT (p. ej. un servidor Mosquitto). Dicha conexión, se realiza por defecto en el puerto 1883 u 8883 en modo seguro [20]. El broker de MQTT recibe los mensajes enviados por un cliente (publicador). Los mensajes son filtrados por medio de topics. Otros clientes (suscriptores) pueden suscribirse a uno o varios topics y el broker les remitirá los mensajes correspondientes.

Cuando un cliente se conecta a un broker MQTT, envía un mensaje de tipo *CONNECT* con su información principal (nombre de usuario, contraseña, ID de cliente, etcétera). El broker replica con un mensaje de tipo *CONNACK* donde especifica el resultado de la conexión donde puede ser aceptada, rechazada u otra (ver figura 3.1) [20].

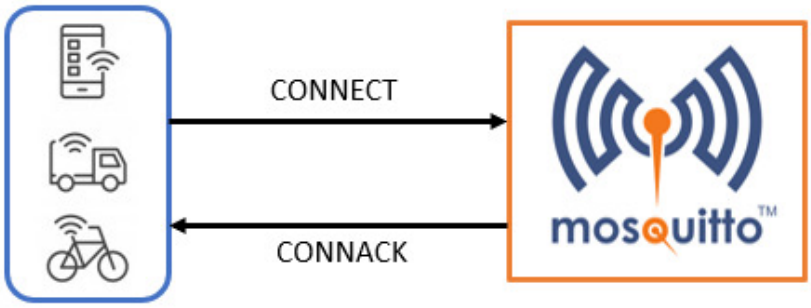


Figura 3.1: Conexión Cliente-Broker MQTT

Una vez que la conexión se establece exitosamente, el cliente envía al broker mensajes de tipo *PUBLISH* con el topic y el contenido del mensaje. De igual forma, se utilizan mensajes de tipo *SUSCRIBE* y *UNSUSCRIBE* para suscribirse o desuscribirse a los distintos topics. El broker responderá a estos mensajes con *SUBACK* y *UNSUBACK* (ver figuras 3.2 y 3.3) [20].

Los clientes envían periódicamente un mensaje de tipo *PINREQ* para asegurar que la conexión con el broker siga activa. El broker avalará esto con un mensaje *PINGRESP*. Si el cliente se desconecta, enviará un mensaje de tipo *DISCONECT* (ver figuras 3.4 y 3.5) [20].

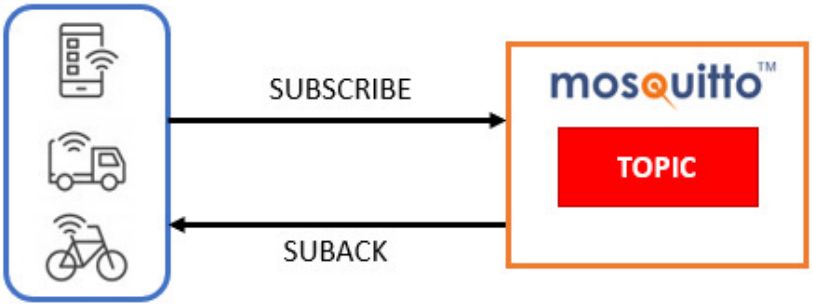


Figura 3.2: Suscripción Cliente-Topic MQTT

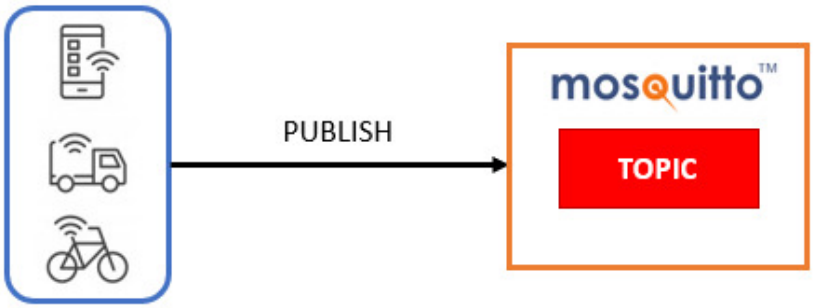


Figura 3.3: Envío de mensajes tipo Publish

```
root@galileo:~# mosquitto_sub -d -h 132.248.32.20 -u alan -P p3ps1c0 -t
 test
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: test, QoS: 0, Options:
0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) sending PINGREQ
Client (null) received PINGRESP
Client (null) received PUBLISH (d0, q0, r0, m0, 'test', ... (17 bytes))
mensaje de prueba
Client (null) sending PINGREQ
Client (null) received PINGRESP
Client (null) sending PINGREQ
Client (null) received PINGRESP
^CClient (null) sending DISCONNECT
root@galileo:~#
```

Figura 3.4: Uso del comando mosquito_sub

Los mensajes en MQTT están estructurados de la siguiente manera:

- **Fixed Header.** Es un código de control obligatorio (de 1 byte) que identifica el tipo y la longitud del mensaje enviado (de 1 a 4 bytes).

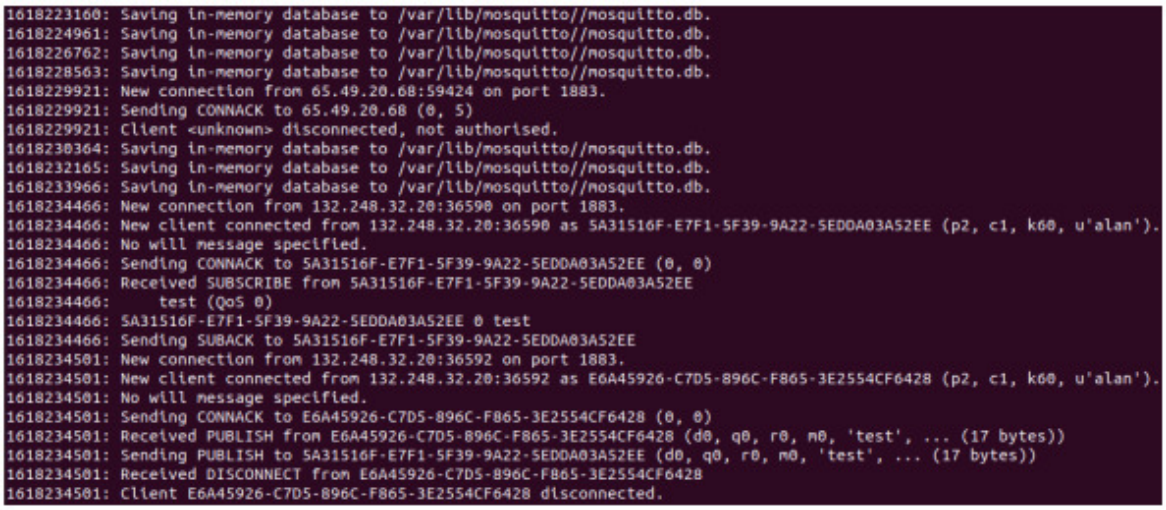


Figura 3.5: Visualización del log de Mosquitto

- **Optional Header.** Contiene información adicional necesaria en ciertos mensajes. Es opcional.
- **Payload.** Es el contenido del mensaje. Su tamaño máximo es de 256 MB.

La **Calidad del Servicio** (Quality of Service, QoS) de MQTT gestiona la robustez en el envío de mensajes ante posibles fallos [20]. Existen tres niveles de QoS:

- **QoS 0 Unacknowledged (At Most One).** El mensaje solo se envía una vez. En caso de fallo, el mensaje se perderá.
- **QoS 1 Acknowledged (At Least One).** El mensaje es enviado hasta garantizar su entrega. Puede existir duplicidad de mensajes.
- **QoS 2 Assured (Exactly One).** Se garantiza la entrega del mensaje una única vez.

La fiabilidad del sistema depende de sus características y necesidades. A mayor nivel de QoS, mayor carga de procesamiento se genera en el dispositivo IoT y el servidor MQTT [20].

La **Seguridad** en MQTT puede establecerse mediante certificados SSL/TLS (Security Sockets Layer/Transport Layer Security) o con autenticación de usuario y contraseña. Generalmente, los dispositivos de IoT tienen poca capacidad de procesamiento, por lo que

usar un certificado SSL/TLS supondría una carga excesiva de trabajo [20]. Por otro lado, la autenticación de usuario y contraseña puede enviarse en texto plano.

3.2.1. Publish/Subscribe (Pub/Sub)

El patrón de mensajería Publish/Subscribe (Pub/Sub) hace de MQTT un protocolo sencillo, ligero, confiable, seguro, escalable, robusto y asíncrono. Ideal para dispositivos de IoT, pues requiere de una capacidad de recursos baja [20].

En dicho modelo, existen dos agentes. El 'Publisher' o publicador, se dedica a enviar mensajes. El 'Suscriber' o suscriptor es aquel que recibe los mensajes. El suscriptor le indica al broker el tipo de mensaje que quiere recibir. El broker distribuye los mensajes a los suscriptores a través de filtros llamados topics. Los topics funcionan de manera sencilla, transparente y jerárquica. Un topic es una cadena de texto (UTF-8) de hasta 65536 caracteres, organizados de manera jerárquica. El nombre de un topic distingue entre mayúsculas y minúsculas. Se pueden crear topics de uno o varios niveles, separados por una diagonal '/' [21]. El ejemplo de un topic de tres niveles es: Flotilla/Taxi/Ubicacion. No hace falta crear topics explícitamente para poder publicar mensajes o suscribirse a ellos. Los publicadores solo pueden enviar mensajes a un único topic. Los suscriptores pueden unirse a uno o varios topics estableciendo distintas suscripciones o usando 'Wildcards' [21]. Existen dos tipos de Wildcards:

- **Wildcard de nivel único.** Se usa el caracter '+' para sustituir un único nivel en cualquier lugar del topic. Ejemplo: Flotilla/+/Ubicacion
  Funciona para:
  Flotilla/Taxi/Ubicacion
  Flotilla/Bus/Ubicacion

- **Wildcard de nivel múltiple.** Se usa el caracter # para sustituir cualquier número de niveles. Solo puede utilizarse al final del topic. Ejemplo: Flotilla/Bus/#
  Recibirá todos los mensajes del topic que comience con Flotilla/Bus

Es recomendable organizar los topics de manera clara, sostenible y con margen de

ampliación (ver figura 3.6).

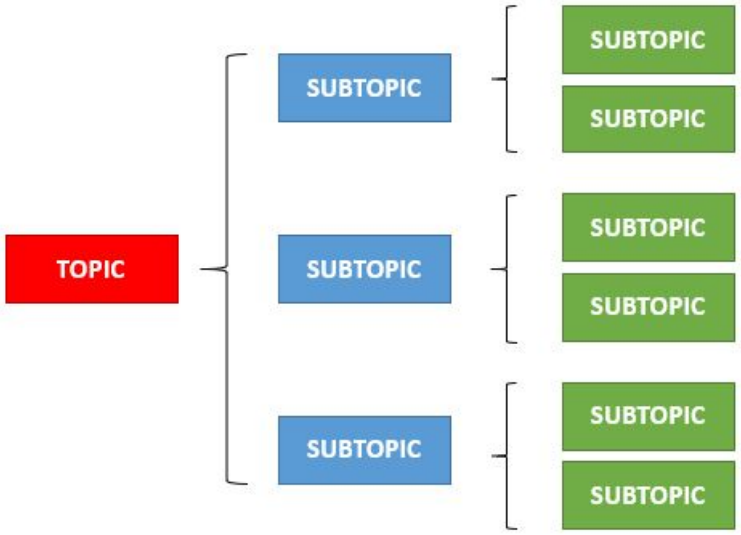


Figura 3.6: Jerarquización de topics

3.2.2. Message Queue (MQ)

MQTT es un protocolo de tipo 'Message Queue' (MQ). Esto significa, que el broker genera una cola de mensajes para cada cliente que se suscribe a un topic [22]. Cada cliente tiene un identificador. El broker usa este identificador para determinar a qué cliente enviar cada mensaje. Los mensajes son recibidos por una MQ para ser entregados al cliente. Si el cliente no está conectado, los mensajes se mantienen en la MQ dentro del broker y serán entregados cuando el cliente se conecte (ver figura 3.7).

Figura 3.7: Cola de mensajes de MQTT

3.2.3. Message Service (MS)

El broker de MQTT cuenta con un servicio de mensajes (Message Service, MS). El MS distribuye de forma inmediata los mensajes a los clientes conectados. A diferencia de MQ, los mensajes enviados se perderán si el cliente está desconectado (ver figura 3.8) [22].

Figura 3.8: Servicio de mensajes de MQTT

3.3. Apache ZooKeeper

Apache ZooKeeper es un servidor de código abierto que ayuda a sincronizar procesos, compartir información de configuración y proporcionar servicios grupales en aplicaciones distribuidas. [23]. Apache Kafka necesita de ZooKeeper para poder coordinar y gestionar sus brokers. ZooKeeper lleva un monitoreo de los cambios realizados en el clúster de Kafka, incluyendo el manejo de fallas. Si se produce algún cambio, por ejemplo, la creación de un topic o el fallo de un broker, envía una notificación a Kafka [24]. ZooKeeper elige la partición líder de cada broker e informa de esto a productores y consumidores. ZooKeeper no almacena offsets de los consumidores, esta tarea la hace Kafka.

Apache Storm también requiere de ZooKeeper para su correcto funcionamiento. Todos los estados, tareas y datos relacionados con el clúster de Storm se almacenan en ZooKeeper [17]. Gracias a esto, se pueden eliminar abruptamente los procesos de Nimbus y los supervisores sin afectar negativamente a Storm. Además, Nimbus y los supervisores de Storm se comunican entre sí a través de ZooKeeper.

Por diseño, el clúster de ZooKeeper opera con un número impar de servidores o nodos; ya sea: 3, 5, 7, etcétera. Esto se recomienda para tener un mayor margen en el manejo de fallas que puedan ocurrir en los nodos del clúster. ZooKeeper nombra un servidor líder que

realiza tareas de escritura. El resto de servidores se denominan 'followers' o seguidores y ejecutan tareas de lectura. Si se cae el servidor líder, se elige uno nuevo entre los nodos restantes (ver figura 3.9) [17].

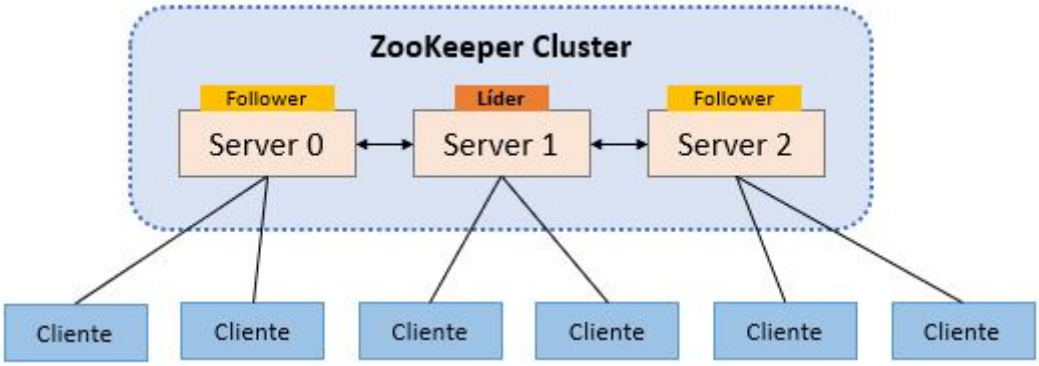


Figura 3.9: Clúster de Apache ZooKeeper

3.4. Apache Kafka

Apache Kafka fue creado por LinkedIn en el año 2011. Actualmente el proyecto es sustentado por la empresa Confluent. Kafka se define como una plataforma de transmisión de eventos distribuidos con capacidad para manejar millones de eventos por segundo [25]. La transmisión de eventos de Kafka se refiere a la captura de datos casi en tiempo real, originarios de fuentes externas como una base de datos, sensores, dispositivos móviles, etcétera. Por lo regular, Kafka es utilizado como mecanismo de transporte de datos. De igual manera, suele ser implementado en sistemas de recolección de mensajes, métricas y bitácoras, actividades de rastreo, sistemas de procesamiento, integraciones con plataformas como Apache Spark, Apache Storm, Apache Flink, Apache Hadoop o cualquier tecnología de Big Data.

Kafka se caracteriza por su escalabilidad horizontal, lo que permite integrar un clúster con cientos de nodos de manera sencilla. Su alta disponibilidad lo hace un sistema tolerante a fallas. Su baja latencia (menor a 10 ms) permite acceder a la información prácticamente en tiempo real. Estas bondades han posicionado a Kafka como una de las tecnologías más usadas por cientos de empresas globales, entre ellas Uber, Netflix, Walmart, Spotify y

Airbnb.

Kafka trabaja bajo el patrón de mensajería Publicación/Suscripción (Pub/Sub). En este modelo, los publicadores crean topics o eventos. Los suscriptores se registran a uno o varios topics. Cada topic puede tener muchos suscriptores. Ellos recibirán una copia de cada mensaje que sea publicado en el topic. En el modelo Pub/Sub, los mensajes pueden ser almacenados en un topic hasta que son enviados a sus suscriptores. Esto se logra gracias al sistema de notificación de mensajes que garantiza la entrega eficiente de todos los eventos publicados en el topic.

3.4.1. Producer

Un 'Producer' o productor de Kafka es el proceso que se encarga de enviar y almacenar datos en los topics. Los productores saben en qué broker y partición del topic deben escribir los datos. Si un broker falla, los productores se recuperarán automáticamente (ver figura 3.10).

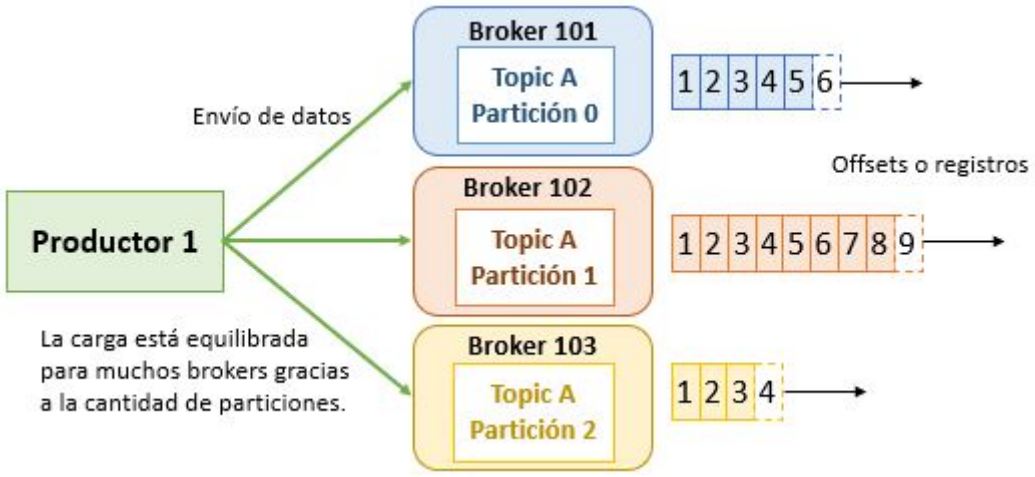


Figura 3.10: Balanceo de carga entre brokers

Los productores se pueden configurar para que obtengan un acuse de recibo (acknowledgment) en la escritura de datos. Estas pueden ser:

- **Acks=0.** El productor no espera ningún acuse. Existe la posibilidad que haya pérdida

de datos.

- **Acks=1.** El productor espera el acuse de la partición líder. Puede haber una pérdida de datos limitada.
- **Acks=all.** El productor espera el acuse de la partición líder y de las réplicas. No existe pérdida de datos.

El productor puede enviar una clave (key) junto con el mensaje. Si se configura la clave como NULL, los datos serán enviados a los brokers mediante el sistema Round Robin. Es decir, primero al broker 101, después al broker 102, luego al broker 103, y así sucesivamente. Pero, si se especifica una 'key', todos los mensajes con la misma clave tendrán como destino la misma partición en el mismo broker. Enviar mensajes con clave es de gran utilidad cuando se necesita ordenarlos u organizarlos en conjuntos. Por ejemplo, si se especifica un campo ID (puede ser un número, un string u otro tipo de dato), todos los mensajes con el mismo ID serán almacenados en la misma partición. Esto se realiza a través de una tabla hash de claves que depende del número de particiones. En la siguiente figura, se muestra que las claves id_123 y id_234 siempre se almacenan en la partición 0 del broker 101, mientras que id_345 y id_456 lo hacen en la partición 1 del broker 102 (ver figura 3.11).

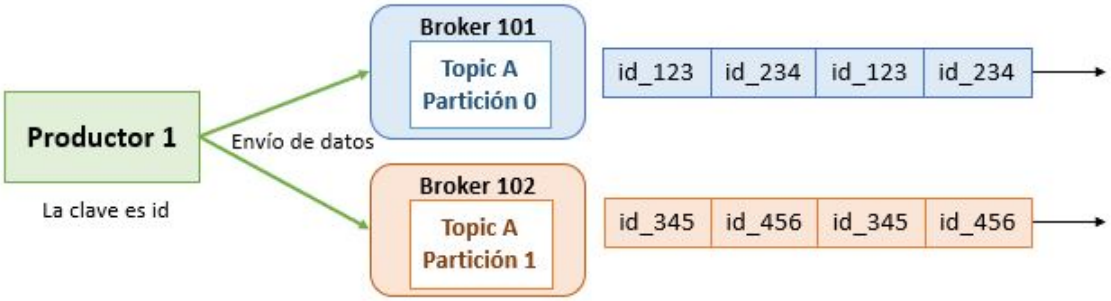


Figura 3.11: Mensajes con clave de un productor

3.4.2. Consumer

Los 'Consumers' o consumidores son procesos que se suscriben a un topic de Kafka y reciben su información. Los consumidores saben de qué broker deben obtener los datos.

Si un broker falla, los consumidores saben cómo recuperarse. Los datos se leen ordenadamente dentro de cada partición pero las particiones se leen en paralelo. Por esto, no se garantiza que primero se lea la partición 0, después la partición 1, enseguida la partición 2 y así sucesivamente.

Los consumidores se pueden agrupar. Incluso, es posible tener varios grupos de consumidores. Un consumidor que pertenece un grupo lee los datos de una partición específica. Por lo tanto, si hay más consumidores que particiones, los consumidores sobrantes estarán inactivos. Por esta razón, se recomienda tener el mismo número de particiones y consumidores. Cuando un consumidor de determinado grupo lee un mensaje de cierta partición, otro consumidor del mismo grupo no podrá volver a leer ese mensaje. Estos procedimientos los realiza Kafka de manera automática a través de un coordinador de grupo (GroupCoordinator) y un coordinador de consumidores (ConsumerCoordinator). En consecuencia, el programador no los puede gestionar ni configurar (ver figura 3.12).

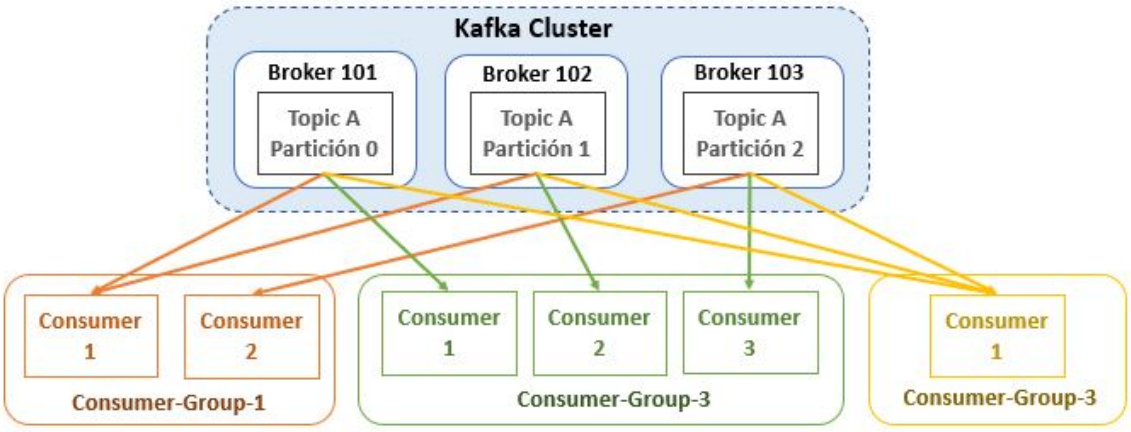


Figura 3.12: Grupos de consumidores accediendo a distintas particiones del Topic A

3.4.3. Componentes de Kafka

Un **clúster de Kafka** se compone de uno o varios nodos llamados **Brokers**. Cada broker se identifican con un número entero y puede almacenar ciertas particiones de un topic. Los brokers de Kafka son *stateless*, por lo que necesitan de un servidor ZooKeeper para mantener el estado del cluster [24]. A través del proceso denominado bootstrap broker, se

establece una conexión con el broker. De esta manera, podemos tener acceso a todos los nodos del clúster (ver figura 3.13).

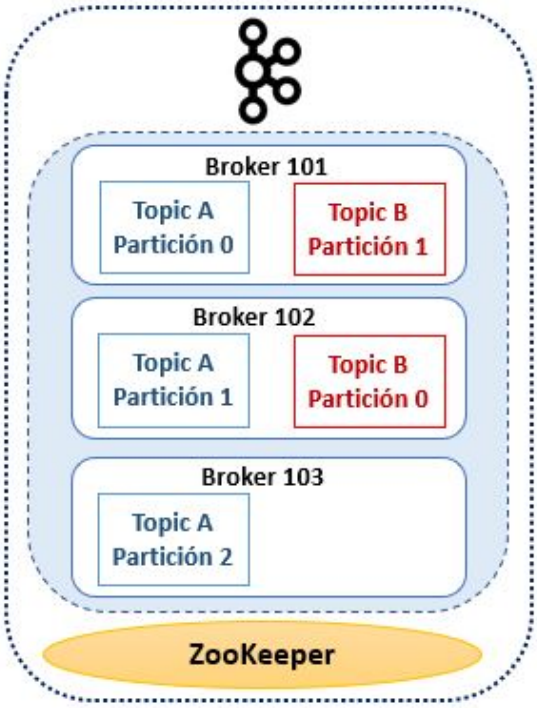


Figura 3.13: Clúster de Apache Kafka

Un **Topic** es una cola de mensajes. También puede definirse como un flujo particular de datos. Se usan para clasificar o categorizar los mensajes enviados a Kafka. Los topics se identifican con un nombre y pueden dividirse en **particiones**. Las particiones de un topic se enumeran de 0 a N y se ordenan de manera ascendente. El número de particiones que tendrá un topic de Kafka se especifican al momento de crearlo y pueden ser modificadas posteriormente de ser necesario. Cada partición almacena mensajes respetando el orden de llegada. Los mensajes contenidos en cada partición se denominan offsets. Un offset es el ID que obtiene cada mensaje dentro de una partición.

En Kafka, la replicación de datos se maneja a nivel de topics. El factor de replicación de los topics se distribuye entre los distintos brokers del clúster. Los topics deben tener un factor de replicación mayor a 1. En la siguiente figura, se ejemplifica el Topic A con dos particiones y un factor de replicación de 2 (ver figura 3.14). Esto quiere decir que, cada

partición del Topic A se replica dos veces en los nodos que componen el clúster. Si el Broker 102 se cae, los Brokers 101 y 103 atenderán la llegada de los datos, sin embargo, estos ya no se podrán replicar. Además, si llegara a caerse otro broker los datos se perderían para siempre. Debido a esto, se recomienda crear los topics con un factor de replicación de 3.

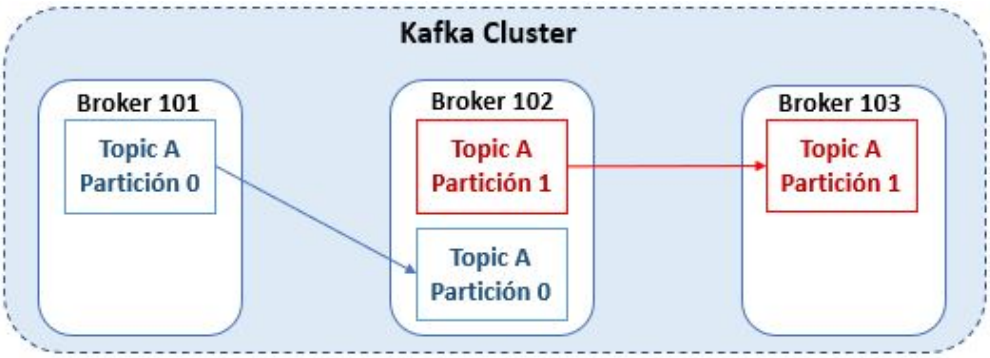


Figura 3.14: Replicación de datos con valor de 2

La replicación de datos requiere de una partición líder (leader partition). La partición líder es la responsable de recibir y entregar los datos. Un broker solo puede tener una partición líder. Los demás brokers del cluster sincronizarán y actualizarán los datos de acuerdo a la partición líder. Por ende, hay una partición líder y muchas particiones in-sync-replica o réplica sincronizada (ISR, por sus siglas en inglés). ZooKeeper se encarga de decidir cuál será la partición líder y cuáles serán las particiones ISR. Si un broker que contiene una partición líder se cae, una partición ISR tomará el rol de líder. Cuando el broker caído vuelva a funcionar retomará la partición líder nuevamente (ver figura 3.15).

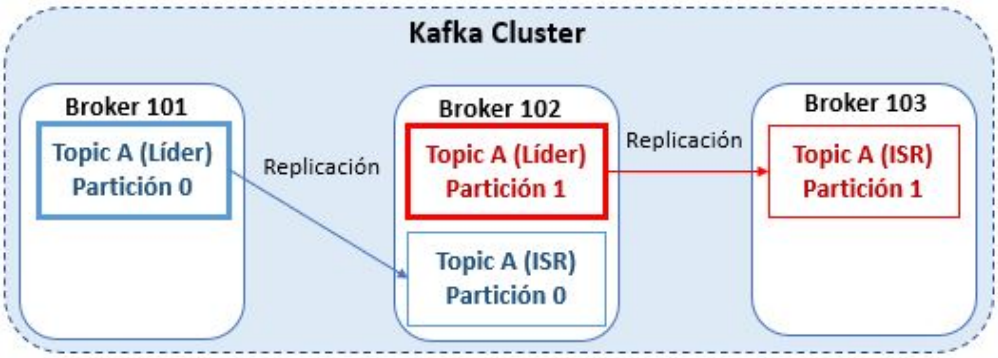


Figura 3.15: Partición líder en Apache Kafka

Un **Offset** es un identificador incremental que contiene una colección de bytes. Esta colección de bytes, son los mensajes provenientes de los productores y se representan como una matriz. El orden de los offsets está garantizado dentro de cada partición, no entre particiones. Cada offset solo tiene un significado para una partición en concreto. Por consiguiente, el offset 1 de la partición 0 no tendrá la misma información que el offset 1 de la partición 1 (ver figura 3.16). Los datos se guardan en Kafka durante un tiempo limitado (7 días, por defecto). Una vez que los datos se escriben en una partición son inmutables, es decir, no se pueden modificar. Los datos se asignan aleatoriamente a una partición, a menos que se indique el ID de la partición.

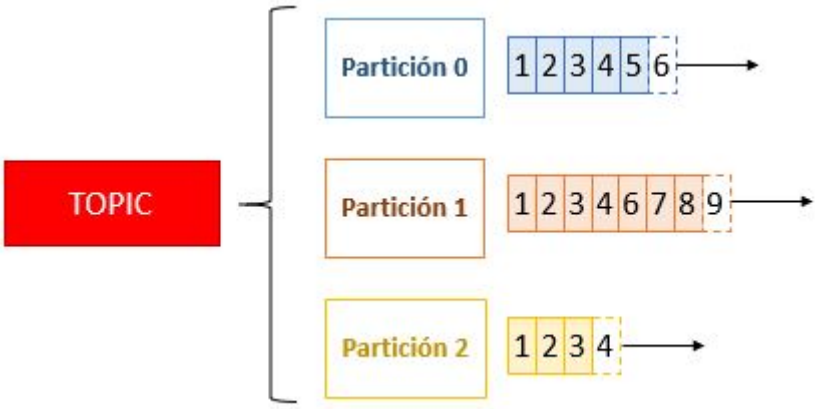


Figura 3.16: Offsets en Apache Kafka

3.5. Apache Storm

Apache Storm es un framework para el procesamiento de flujos de datos distribuido, en tiempo real, confiable y tolerante a fallas. Storm tiene la capacidad de procesar eventos con millones de registros por segundo. Generalmente, Storm es implementado para el manejo de aplicaciones sensibles al tiempo, donde el retraso de unos pocos segundos puede ocasionar grandes pérdidas de datos [26]. Algunos casos de uso de Storm son: analítica en tiempo real, creación de motores de recomendación, detección de fraudes, activación de actividades sospechosas, paralelización de operaciones con uso intensivo de recursos de cómputo, entre otros.

Storm trabaja bajo el modelo Maestro-esclavo. Un clúster de Storm está integrado por un nodo maestro y muchos nodos esclavos o trabajadores (ver figura 3.17). Nimbus es el proceso que levanta un nodo maestro. Supervisor es el proceso que crea un nodo trabajador. Storm es un sistema *stateless*, por lo que necesita un cluster de Apache ZooKeeper para su pleno funcionamiento. ZooKeeper coordinará y guardará los datos de configuración de Nimbus y los Supervisores. Por tal motivo, si los nodos de Storm fallan o se reinician, no afectan el estado general del sistema. Otra cualidad de Storm, es su simplicidad en la programación y el desarrollo de aplicaciones. Se puede desarrollar en cualquier lenguaje de programación que lea y escriba flujos de datos de entrada y salida (stardard input/stardard output, stdin/stdout) [17].

Es posible monitorear los diversos procesos de un clúster de Storm a través de su interfaz de usuario (User Interface, UI). La UI de Storm se ejecuta de manera predeterminada en el puerto 8080 del equipo. Podemos acceder a ella desde un navegador web, ingresando la dirección IP (Internet Protocol) de la máquina y el puerto indicado.

3.5.1. Componentes de Storm

- **Nimbus.** Es el nodo maestro en un cluster de Storm. Hay un único nodo activo de Nimbus en el cluster pero puede existir un nodo pasivo. El nodo pasivo se convertirá en activo si este falla. Las funciones del nodo maestro consisten en distribuir los datos y asignar las tareas a los nodos trabajadores, monitorear fallas y recuperarse de ellas si se presentan. Si un nodo trabajador falla, Nimbus reasigna sus tareas a otro nodo trabajador.

- **Supervisor.** Es el nodo trabajador en un cluster de Storm. Puede haber múltiples Supervisores. Son los responsables de crear, iniciar y detener los procesos de trabajo que ejecuten las tareas asignadas por Nimbus. Cada Supervisor tiene uno o varios procesos de trabajo. Los nodos trabajadores pueden funcionar incluso si todos los nodos de Nimbus fallan. Sin embargo, no se podrán añadir nuevas tareas.

- **Tupla.** Es un registro o mensaje. La tupla es la unidad básica de datos que procesa Storm. Cada tupla está constituida por una lista de campos. Los campos pueden tomar el valor de algún tipo de dato (char, integer, long, float, booleano, byte, matriz de bytes). Inclusive, se permite definir tipos de datos propios usando la API de Storm. Esto facilita la serialización de los campos de una tupla.
- **Stream.** Es una secuencia ilimitada de tuplas. Cada stream tendrá su propio ID. Storm procesa los streams en paralelo.
- **Spout.** Es el agente que lee los datos de una fuente externa y los envía a los bolts en forma de streams. Un spout puede emitir uno o múltiples streams.
- **Bolt.** Son las unidades de procesamiento de Storm. Un bolt puede procesar uno o varios streams.

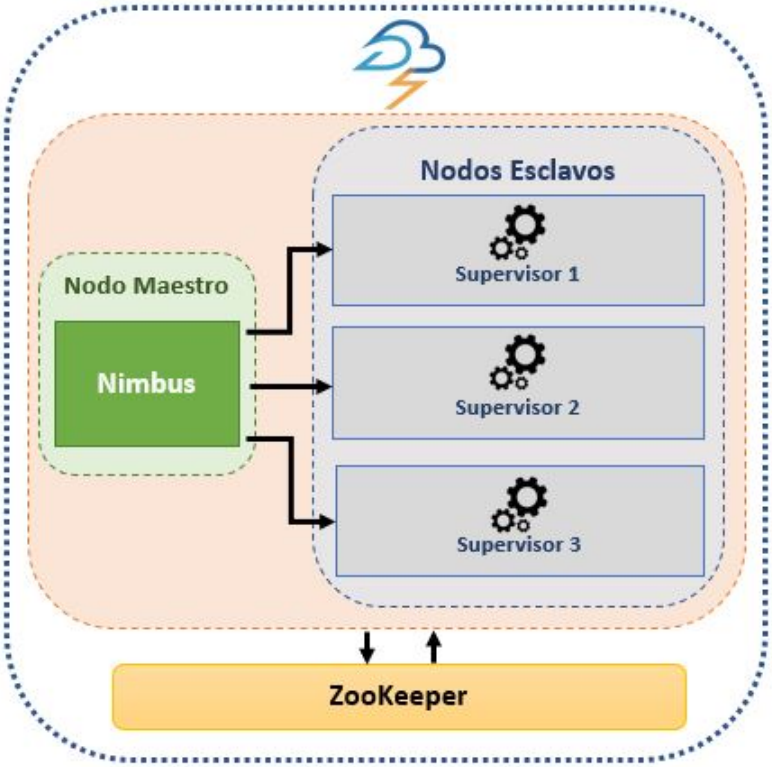


Figura 3.17: Clúster de Apache Storm

3.5.2. Spout

La topología de Storm está conformada por Spouts y Bolts. Spout es el componente utilizado para leer los flujos de datos de un sistema de origen externo. La manera de emitir datos o registros desde Spout es a través de tuplas. Una tupla es un valor serializable con uno o varios campos de cualquier tipo. Las tuplas se agrupan en streams. Cada stream tiene un identificador y un esquema propio. Los esquemas de los streams se refieren al número de campos que contendrán las tuplas que transiten por ellos. Spout tiene la capacidad de producir varios streams con diferentes esquemas cada uno.

Existen dos maneras de administrar el procesamiento de los datos en Spout:

- **Spout confiable.** Se espera un acuse de recibo de cada registro para su posterior procesamiento. Si ocurren fallas en el procesamiento de los datos, estos se vuelven a producir. La configuración de un Spout confiable requiere de más recursos de cómputo pero garantiza la lectura de todos los registros al menos una vez.
- **Spout no confiable.** No espera ningún acuse de recibo y no vuelve a emitir los datos en caso de falla. Consume menos recursos de cómputo y es ideal para aplicaciones donde la pérdida de datos no afecta el rendimiento del sistema.

3.5.3. Bolt

Bolt es el componente que recibe y procesa los datos contenidos en las tuplas que fluyen a través de los streams. Un Bolt puede suscribirse a uno o múltiples streams. Los streams pueden provenir de un Spout o de otros Bolts. Los Bolts pueden emitir uno o múltiples streams de salida. La transformación de los datos puede ser simple o compleja. Si es simple, la tarea la realiza un solo Bolt. Si es compleja, se coordinan varios Bolts para ejecutar el trabajo. Una vez procesados los registros, Bolt puede almacenarlos en una base de datos o cualquier sistema de persistencia de datos (ver figura 3.18).

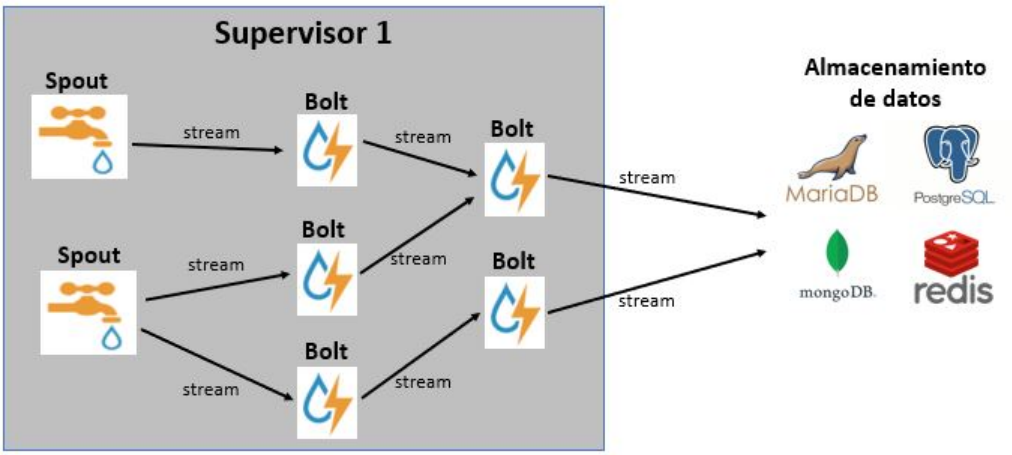


Figura 3.18: Topología de un nodo trabajador

3.6. Persistencia de datos

La persistencia de datos es la acción de guardar la información generada en un sistema de almacenamiento, para su posterior utilización [27]. Al sistema de almacenamiento, también se le conoce como fuente de datos (Data Source). Estas pueden ser ficheros, bases de datos u otras.

3.6.1. MySQL

MySQL es un sistema gestor de bases de datos relacional (Relational Database Management System, RDBMS), rápido, robusto, multiusuario, multiproceso y multiplataforma [28]. Como su nombre lo indica, el servidor de MySQL emplea el Lenguaje de Consulta Estructurado (Structured Query Language, SQL) en su operatividad. Este lenguaje, se usa para el manejo de la información en la base de datos. MySQL tiene dos licencias: la primera, es de código abierto; la segunda, es una licencia comercial de Oracle.

Estas son algunas de las principales características de MySQL [29]:

- **Arquitectura Cliente-servidor.** MySQL soporta la conexión de varios clientes al servidor sin afectar el rendimiento. Cada cliente puede realizar consultas para insertar, eliminar, modificar o recuperar registros en las bases de datos.

- **Vistas.** Se denominan tablas virtuales. Ya que, su único propósito es el de mostrar información de tablas reales que sí existen en la base de datos.
- **Transacciones.** Son una sucesión de pasos que actúan como una sola operación. Se tienen que cumplir con éxito todas las fases para que el proceso se lleve a cabo. De lo contrario, la operación fallará.
- **Desencadenantes.** Se refiere a la posibilidad de automatizar tareas dentro de una base de datos.
- **Procedimientos almacenados.** Son conjuntos de comandos SQL que pueden almacenarse en el servidor. Esto se usa para que los clientes no ejecuten comandos uno por uno. Adicionalmente, crea un entorno más fuerte y seguro [30].

Capítulo 4

ARQUITECTURA ESCALABLE DISTRIBUIDA

Esta sección describe el diseño de la arquitectura escalable distribuida de código abierto propuesta para el seguimiento de geolocalización en tiempo real.

4.1. Diseño de la arquitectura distribuida para el procesamiento de datos de geolocalización en tiempo real

El diseño y la arquitectura se dividen en cuatro etapas. La primera etapa utiliza una marca de tiempo GPS correspondiente a las coordenadas de ubicación de un vehículo en movimiento. La marca de tiempo contiene cuatro datos: ID del vehículo, fecha y hora, coordenadas de latitud y longitud. Los datos se recopilan desde un dispositivo IoT que envía marcas de tiempo cada intervalo de tiempo a un servidor MQTT. Los componentes del dispositivo IoT utilizados en el diseño de la plataforma son una placa Arduino MEGA y un módulo GSM/SIM808. La segunda etapa clasifica la información por temas usando Kafka. La tercera etapa calcula las marcas de tiempo utilizando una topología en Apache Storm. En esta etapa, las cadenas de marcas de tiempo se dividen por comas. Cada parte de la marca de tiempo se envía a otro proceso Bolt para su procesamiento. En la última etapa, la persistencia de datos almacena las marcas de tiempo del GPS directamente en una base de datos para futuros análisis. La figura 4.1 muestra la arquitectura de la plataforma de big

data de código abierto propuesta para el seguimiento de geolocalización en tiempo real.

La arquitectura escalable distribuida de código abierto propuesta para el seguimiento de geolocalización en tiempo real se implementa utilizando MQTT, Kafka, Storm y MySQL. Un dispositivo IoT realiza la recopilación de datos a través de una placa Arduino MEGA y un módulo GSM/SIM808. El agente MQTT está disponible mediante un servidor Mosquitto en un host en la nube con una IP pública. El agente filtra los mensajes entrantes del cliente del dispositivo IoT en un topic. Además, se instala un servidor Zookeeper para coordinar la comunicación del proceso entre los nodos del clúster. La comunicación entre Mosquitto y Kafka utiliza el complemento Kafka connect de Confluent para enviar y recibir datos del topic de MQTT. Luego, se crea una partición de Kafka con un factor de replicación uno en el agente de Kafka para recopilar datos en un topic de Kafka. Una vez que los agentes de MQTT y Kafka están funcionando, la topología de Storm se implementa en el clúster de Storm.

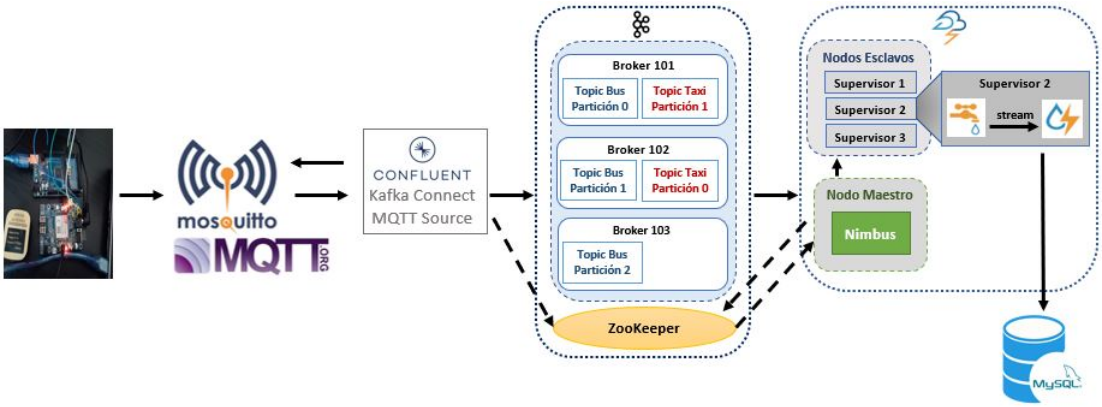


Figura 4.1: Diseño de la arquitectura escalable distribuida

Storm funciona en modo local y en modo clúster (es decir, modelo maestro-esclavo). Los procesos de Storm se coordinan a través del servicio Zookeeper. Storm tiene un nodo maestro llamado Nimbus y al menos un nodo trabajador (Supervisor). Los datos de entrada son enviados por el elemento Spout que transforma los datos en tuplas (flujos de datos). El elemento Bolt ejecuta operadores o funciones sobre las tuplas. La topología define el flujo de flujos de datos entre Spouts y Bolts mediante un gráfico acíclico dirigido. La topología se

usa para implementar el modelo paralelo computacional en el clúster de Storm.

La figura 4.2 muestra el flujo de trabajo de boquilla/perno en la topología propuesta. Las flechas azules representan la fase de chorro y las flechas verdes representan la fase de cerrojo. Cada marca de tiempo de GPS se recopila mediante el componente kafkaSpout de un tema de Kafka, y el flujo de datos (tupla) se envía a diferentes pernos (SplitBolt y MySQLBolt). Por lo tanto, cada perno realiza una tarea diferente. Primero, SplitBolt recibe una tupla que la almacena en una variable de cadena. Después de eso, la cadena se divide por comas y se envían dos nuevas tuplas a CountBolt. Luego, CountBolt clasifica las tuplas entrantes para acumular las distancias de las trayectorias. Una vez que finaliza la operación de conteo, una nueva tupla envía una nueva tupla con un valor de distancia al ReportBolt y almacena las distancias procesadas en la memoria para mostrarlas en un informe final. Por otro lado, MySQLBolt realiza la persistencia de los datos, almacenando la marca de tiempo del GPS directamente en una base de datos MySQL.

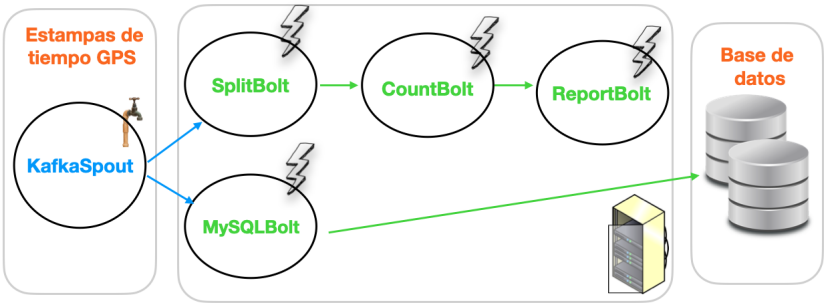


Figura 4.2: Topología Storm para seguimiento de geolocalización en tiempo real

4.2. Integración del dispositivo IoT

Para la generación de los datos de geolocalización GPS en tiempo real, se emplearon los siguientes elementos de hardware:

- Placa de Arduino Mega 2560
- Módulo SIM808

- Tarjeta SIM (Subscriber Identity Module) con acceso a la red móvil
- Antena GSM
- Antena GPS estándar
- Fuente de alimentación 9 Voltios, 2 Amperes

La figura 4.3 muestra los distintos componentes de hardware usados por el dispotivo IoT.

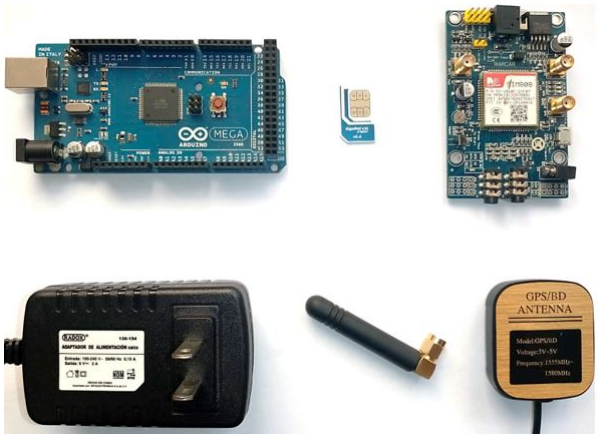


Figura 4.3: Componentes de hardware del dispositivo IoT

Para la instalación e implementación de los brokers de MQTT Mosquitto, Apache Kafka, Apache Storm, Apache ZooKeeper y el servidor de base de datos MySQL es necesario: un usuario con permisos de administrador (root), tener instalado JDK (Java Developer Kit) 8 y Python versión 2.6 o 2.7. Además, para la creación y despliegue de los programas se dispuso de Arduino IDE (Integrated Development Environment) 1.8.3, Apache NetBeans 12.0 y Apache Maven versión 3.3.

La figura 4.4 muestra el esquema de configuración del hardware donde se conectan las antenas GPS y GMS al módulo SIM808.

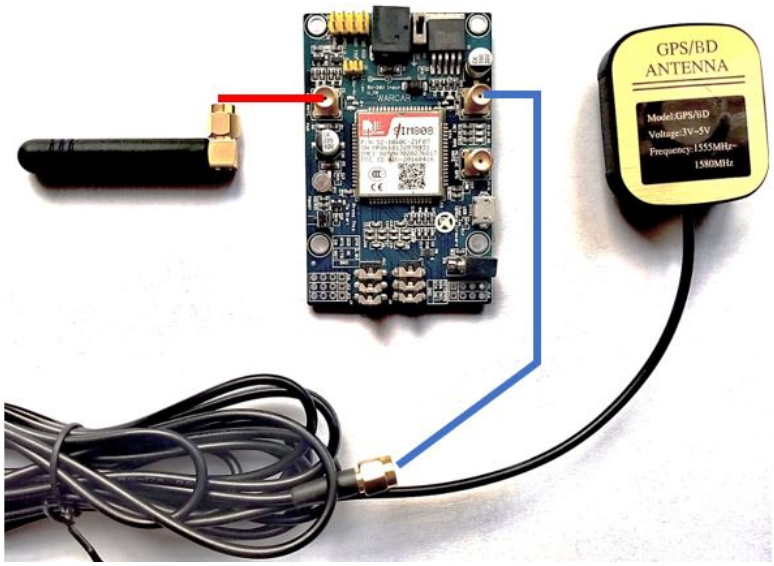


Figura 4.4: Conexión módulo SIM808 con antenas GPS y GSM

La configuración de pines digitales se efectúa usando las siguientes conexiones entre la placa de Arduino y el módulo SIM808:

- Arduino GND a SIM808 GND.
- Arduino TX3 (pin 14) a SIM808 RDX.
- Arduino RX3 (pin 15) a SIM808 TXD.

La figura 4.5 muestra el esquema de conexión entre la entre placa Arduino y el módulo SIM808.

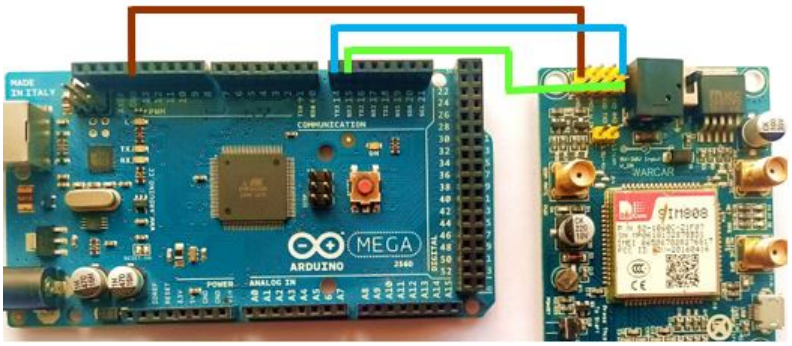


Figura 4.5: Conexión entre placa Arduino y módulo SIM808

Finalmente, se conecta la fuente de alimentación al módulo SIM808. La placa de Arduino se enciende a través de su cable USB mediante un equipo de cómputo para cargar el código del programa que genera los datos de geolocalización en periodos de tiempo específicos de manera constante. La figura 4.6 muestra el prototipo final del dispositivo IoT funcionando.

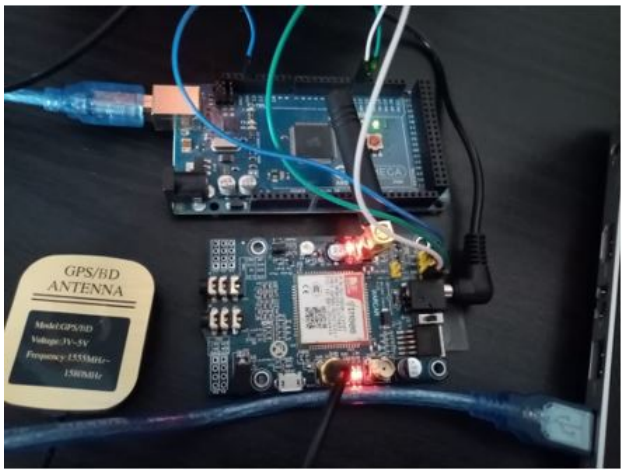


Figura 4.6: Módulo de geolocalización en funcionamiento

4.3. Integración de MQTT con Kafka

El protocolo MQTT se ha convertido en un estándar dentro del paradigma de Internet de las Cosas por ser liviano, flexible y asíncrono [31]. Por estas características, ha sido el elegido para recoger los datos GPS generados por el dispositivo IoT. A su vez, existen una gran cantidad de brokers MQTT en el mercado. En este trabajo, se optó por el broker MQTT Mosquitto. Mosquitto fue desarrollado por Eclipse Foundation, es de código abierto, ligero e ideal para equipos con recursos de cómputo limitados [32].

Para instalar el broker MQTT Mosquitto se debe iniciar sesión como usuario administrador (root). Se procede actualizar las paqueterías del sistema y agregar el repositorio de Mosquitto. Se deben instalar los paquetes Mosquitto y Mosquitto-clients. Mosquitto establece el broker MQTT y Mosquitto-clients es el mecanismo que entabla comunicación con el broker. Hecho esto, se debe editar el archivo de configuraciones de Mosquitto con los parámetros que se muestran en la figura 4.7.

```
  GNU nano 4.8          /etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

persistence true
persistence_location /var/lib/mosquitto/
allow_anonymous false

password_file /etc/mosquitto/passwd
log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 1883
# Save all log in file
#log_dest file /var/log/mosquitto/mosquitto.log
log_type all
log_timestamp true


acl_file /etc/mosquitto/aclfile
```

Figura 4.7: Archivo de configuración de Mosquitto

Para tener acceso al broker de Mosquitto, se crea un usuario con el siguiente comando: *mosquitto_passwd -c /etc/mosquitto/passwd $User*. Posteriormente, se ingresa y se confirma una contraseña segura. A continuación, se crea la lista de control de acceso (Access Control List, ACL) en la ruta que se indicó en el archivo de configuración de Mosquitto. La figura 4.8 muestra los criterios a editar de la ACL.

```
  GNU nano 4.8                /etc/mosquitto/aclfile
user alan
topic readwrite #
topic read $SYS/#
```

Figura 4.8: Lista de control de accesso (ACL) de Mosquitto

En consecuencia, el usuario especificado puede publicar y suscribirse a todos los topics del broker. Con esto, se finaliza la instalación y configuración de Mosquitto. Para guardar los cambios, reiniciar el servicio de Mosquitto con el comando: *systemctl restart mosquitto*. Posteriormente, se comprueba que el demonio de Mosquitto esté activo y en ejecución con el comando: *systemctl status mosquitto* (ver figura 4.9).

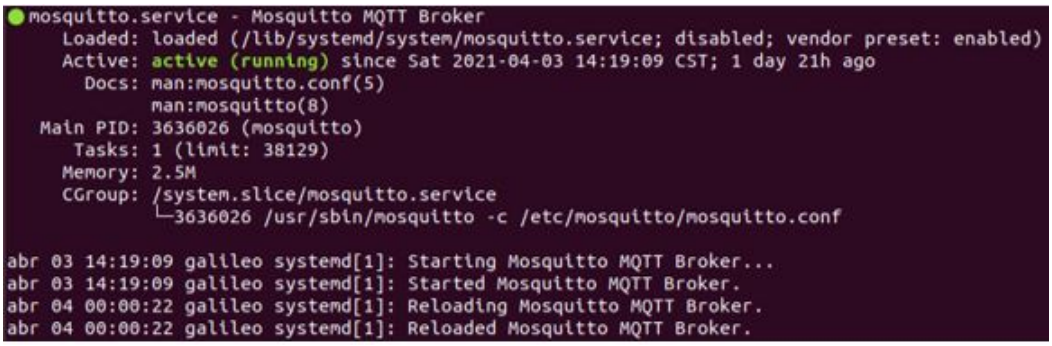


Figura 4.9: Servicio activo de Mosquitto

Si el servicio de Mosquitto está inactivo, se debe habilitar con los comandos: *systemctl enable mosquitto* y *systemctl start mosquitto*.

A continuación se coteja que el puerto predeterminado de Mosquitto (1883) esté escuchando con el comando: *lsof -i -P -n — grep LISTEN* (ver figura 4.10).

Figura 4.10: Puerto 1883 activo de Mosquitto

Una vez que el broker de Mosquitto está habilitado, se procede a hacer una prueba de funcionamiento. En una terminal, se crea un suscriptor con el comando: *mosquitto_sub -d -h 127.0.0.1 -u $User -P ******* -t test* (ver figura 4.11)

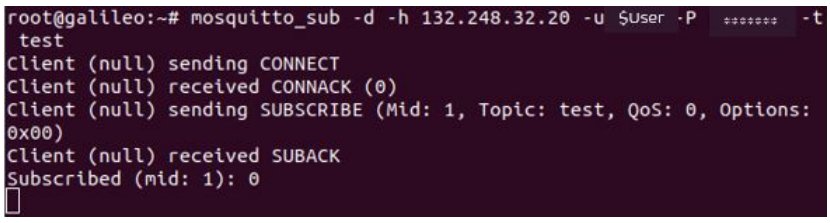


Figura 4.11: Ejemplo comando mosquitto_sub

En una terminal distinta, se crea un productor que envíe un mensaje al mismo topic del suscriptor. Se usa el comando: *mosquitto_pub -d -h 127.0.0.1 -u $User -P ******* -t test -m "mensaje de prueba"* (ver figura 4.12).

```
root@galileo:~# mosquitto_pub -d -h 132.248.32.20 -u $User -P *******  -t
test -m "mensaje de prueba"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q0, r0, m1, 'test', ... (17 bytes))
Client (null) sending DISCONNECT
root@galileo:~#
```

Figura 4.12: Ejemplo comando mosquitto_pub

El siguiente mensaje debe aparecer en la terminal del suscriptor (ver figura 4.13).

```
root@galileo:~# mosquitto_sub -d -h 132.248.32.20 -u $User -P *******  -t
 test
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: test, QoS: 0, Options:
0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) sending PINGREQ
Client (null) received PINGRESP
Client (null) received PUBLISH (d0, q0, r0, m0, 'test', ... (17 bytes))
mensaje de prueba
```

Figura 4.13: Suscriptor recibe mensaje enviado por el publicador

Posteriormente, se usa el comando *tail -f /var/log/mosquitto/mosquitto.log* para acceder a la bitácora (logs) del broker Mosquitto para monitorearlo en tiempo real. La figura 4.14 muestra un ejemplo de la salida de los logs de Mosquitto.

```
1618223160: Saving in-memory database to /var/lib/mosquitto//mosquitto.db.
1618224961: Saving in-memory database to /var/lib/mosquitto//mosquitto.db.
1618226762: Saving in-memory database to /var/lib/mosquitto//mosquitto.db.
1618228563: Saving in-memory database to /var/lib/mosquitto//mosquitto.db.
1618229921: New connection from 65.49.20.68:59424 on port 1883.
1618229921: Sending CONNACK to 65.49.20.68 (0, 5)
1618229921: Client <unknown> disconnected, not authorised.
1618230364: Saving in-memory database to /var/lib/mosquitto//mosquitto.db.
1618232165: Saving in-memory database to /var/lib/mosquitto//mosquitto.db.
1618233966: Saving in-memory database to /var/lib/mosquitto//mosquitto.db.
1618234466: New connection from 132.248.32.20:36590 on port 1883.
1618234466: New client connected from 132.248.32.20:36590 as SA31516F-E7F1-5F39-9A22-5EDDA03A52EE (p2, c1, k60, u'alan').
1618234466: No will message specified.
1618234466: Sending CONNACK to SA31516F-E7F1-5F39-9A22-5EDDA03A52EE (0, 0)
1618234466: Received SUBSCRIBE from SA31516F-E7F1-5F39-9A22-5EDDA03A52EE
1618234466:     test (QoS 0)
1618234466: SA31516F-E7F1-5F39-9A22-5EDDA03A52EE 0 test
1618234466: Sending SUBACK to SA31516F-E7F1-5F39-9A22-5EDDA03A52EE
1618234501: New connection from 132.248.32.20:36592 on port 1883.
1618234501: New client connected from 132.248.32.20:36592 as E6A45926-C7D5-896C-F865-3E2554CF6428 (p2, c1, k60, u'alan').
1618234501: No will message specified.
1618234501: Sending CONNACK to E6A45926-C7D5-896C-F865-3E2554CF6428 (0, 0)
1618234501: Received PUBLISH from E6A45926-C7D5-896C-F865-3E2554CF6428 (d0, q0, r0, m0, 'test', ... (17 bytes))
1618234501: Sending PUBLISH to SA31516F-E7F1-5F39-9A22-5EDDA03A52EE (d0, q0, r0, m0, 'test', ... (17 bytes))
1618234501: Received DISCONNECT from E6A45926-C7D5-896C-F865-3E2554CF6428
1618234501: Client E6A45926-C7D5-896C-F865-3E2554CF6428 disconnected.
```

Figura 4.14: Log de Mosquitto

Posteriormente, se procede a realizar la instalación de Apache Kafka. Para un rendimiento ideal del broker de Kafka, es importante instalar el servidor de Apache ZooKeeper. Para la instalación de ZooKeeper, abrir un navegador web e ir al sitio oficial de descargas

https://zookeeper.apache.org/releases.html. Se da clic en la última versión estable, en este caso, **Apache ZooKeeper 3.7.0(asc, sha512)** y se descarga el archivo desde el enlace proporcionado. Una vez descargado el archivo, se descomprime el archivo y se renombra la carpeta como *zookeeper*. A continuación se ingresa al directorio de configuración de ZooKeeper (*conf*) y se renombra el archivo *zoo_sample.cfg* por *zoo.cfg*. Posteriormente, se edita el archivo *zoo.cfg*, con la configuración que se muestra en la figura 4.15.

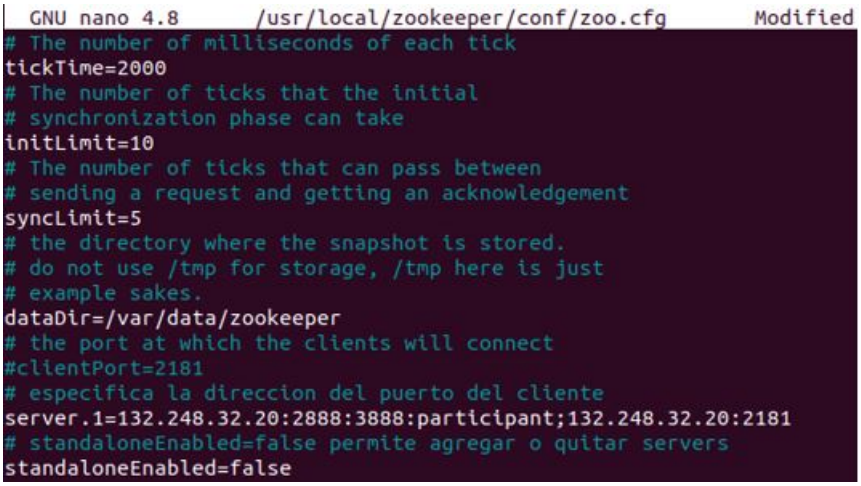


Figura 4.15: Archivo de configuración de ZooKeeper

En este paso, se crea el directorio de datos de ZooKeeper (*dataDir*), tal y como se definió en el archivo de configuración. En el mismo directorio, se crea un archivo llamado *myid*. Una vez creado el directorio, se agrega al archivo *myid* el número de servidor ZooKeeper, en este caso es el servidor 1 (ver figura 4.16).

Figura 4.16: Archivo myid

Una vez finalizado el proceso, se hace una prueba de funcionamiento de ZooKeeper. En una terminal, dentro del directorio bin de ZooKeeper, se introduce el siguiente comando para iniciar el servidor: *./zkServer.sh start*. La figura 4.17 muestra un ejemplo del mensaje de salida.

```
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

Figura 4.17: Servidor ZooKeeper inicializado

El estatus del servidor ZooKeeper se puede conocer desde el directorio bin de ZooKeeper, utilizando el parámetro *status* en el comando: *./zkServer.sh status*. La figura 4.18 muestra la salida del comando anterior.

```
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

Figura 4.18: Estatus del servidor ZooKeeper

Para visualizar el demonio de ZooKeeper (QuorumPeerMain) ejecutándose en la máquina virtual de Java (JVM) se usa el comando: *jps*. La figura 4.19 muestra la salida del comando ejecutado.

```
4093122 QuorumPeerMain
4093216 Jps
```

Figura 4.19: ZooKeeper corriendo en la JVM

Una vez instalado y configurado ZooKeeper, se continua con la instalación del broker de Kafka. En un navegador web, se ingresa al sitio oficial de descargas de Apache Kafka https://kafka.apache.org/downloads. posteriormente, se descarga la última versión estable recomendada, en este caso es la versión **Scala 2.13 kafka_2.132.7.0.tgz (asc, sha512)**. Una vez finalizada la descarga, se descomprime el archivo y se renombra como *kafka*. Posteriormente, se ingresa al directorio de configuración de Kafka (*config*) y se modifican los parámetros que se muestran en la figura 4.20 en el archivo *server.properties*.

```
############################# Log Basics #############################

# A comma separated list of directories under which to store log files
log.dirs=/var/data/kafka

# root directory for all kafka znodes.
zookeeper.connect=132.248.32.20:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=20000
```

Figura 4.20: Configuración archivo server.properties

Una vez modificados los parámetros por default de kafka, se crea el directorio donde se almacenaran los logs de Kafka (*log.dirs*). La ruta del directorio se especifica dentro del archivo *server.properties*. Para levantar el broker de Kafka, primero, se inicializa el servidor ZooKeeper en una terminal. En otra terminal, se accede al directorio bin de Kafka y se ejecuta el siguiente comando: *./kafka-server-start.sh ../config/server.properties*. La figura 4.21 muestra la salida de que el servicio ha sido iniciado.

```
[2021-04-14 04:29:49,609] INFO [SocketServer brokerId=0] Starting socket server acceptors and processors (kafka.network.SocketServer)
[2021-04-14 04:29:49,613] INFO [SocketServer brokerId=0] Started data-plane acceptor and processor(s) for endpoint : ListenerName(PLAINTEXT) (
kafka.network.SocketServer)
[2021-04-14 04:29:49,613] INFO [SocketServer brokerId=0] Started socket server acceptors and processors (kafka.network.SocketServer)
[2021-04-14 04:29:49,615] INFO Kafka version: 2.7.0 (org.apache.kafka.common.utils.AppInfoParser)
[2021-04-14 04:29:49,615] INFO Kafka commitId: 448719dc99a19793 (org.apache.kafka.common.utils.AppInfoParser)
[2021-04-14 04:29:49,615] INFO Kafka startTimeMs: 1618392589613 (org.apache.kafka.common.utils.AppInfoParser)
[2021-04-14 04:29:49,616] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
[2021-04-14 04:29:49,658] INFO [broker-0-to-controller-send-thread]: Recorded new controller, from now on will use broker 0 (kafka.server.Brok
erToControllerRequestThread)
```

Figura 4.21: Broker de Kafka inicializado

Para revisar el correcto despliegue del broker de Kafka, en una nueva terminal, se accede al directorio bin de Kafka y se crea un topic de prueba (aquí nombrado *testKafka*) con el comando: *./kafka-topics.sh --zookeeper 127.0.0.1:2181 --topic testKafka --create --partitions 1 --replication-factor 1*. La figura 4.22 muestra la salida del comando.

```
root@galileo:/usr/local/kafka/bin# kafka-topics.sh --zookeeper 132.248.32.20:2181
--topic testKafka --create --partitions 1 --replication-factor 1
Created topic testKafka.
root@galileo:/usr/local/kafka/bin#
```

Figura 4.22: Comando para crear un topic de Kafka

Posteriormente, se abre una terminal para crear un productor. Para crear el productor se accede a la carpeta bin de Kafka y se ejecuta el comando: *./kafka-console-producer.sh --broker-list 127.0.0.1:9092 --topic testKafka*. Una vez ejecutado el comando, aparece el

símbolo > (mayor que) en el prompt donde se podrá enviar una serie de mensajes. La figura 4.23 muestra la manera de enviar un mensaje desde la terminal a un productor de Kafka.

Figura 4.23: Enviar mensajes desde un productor de Kafka

Para leer los mensajes expedidos por el productor en el topic *testKafka*, se abre una nueva terminal para crear un consumidor usando el siguiente comando: *./kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic testKafka*. La figura 4.24 muestra que el consumidor recibe los datos enviados por el productor.

Figura 4.24: Consumidor lee los mensajes del topic

Para comenzar con la integración de MQTT Mosquitto y Kafka, se accede al directorio de conectores distribuidos ubicado en el directorio de configuración de Kafka. Posteriormente, se configuran los criterios que se muestran en la figura 4.25 dentro del archivo *connect-distributed.properties*.

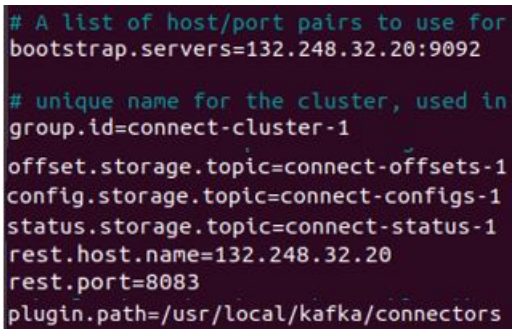


Figura 4.25: Archivo connect-distributed.properties

Una vez dentro de la carpeta, se crea el directorio *connectors* en la ruta que se indicó dentro del archivo *connect-distributed.properties*. Luego, en un navegador web, se accede al sitio oficial del **Conector MQTT (Source and Sink)** para Kafka: https://www.confluent.io/hub/confluentinc/kafka-connect-mqt. En el sitio web, se descarga el archivo comprimido y se reubica en el directorio *connectors* de Kafka. Una vez descargado el archivo se procede a descomprimir el conector y se renombra como *mqtt-source*.

El siguiente paso es configurar el conector *mqtt-source*. Por lo tanto, se inician los servicios de ZooKeeper y Kafka. Asimismo, se inicia una conexión distribuida de Kafka. Para tal efecto, se accede desde una terminal al directorio bin de Kafka y se ejecuta el comando siguiente (ver figura 4.26): *./connect-distributed.sh ../config/connect-distributed.properties*

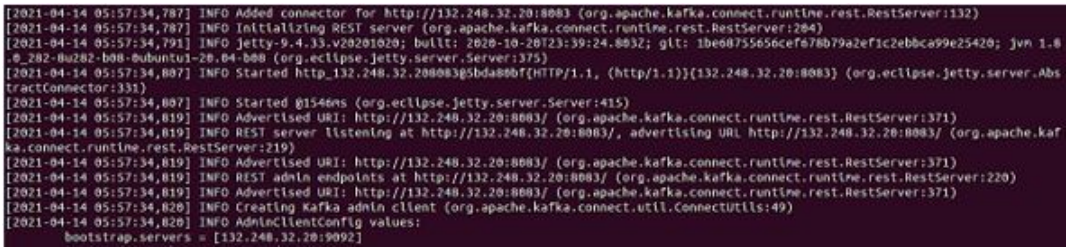


Figura 4.26: Conector distribuido de Kafka en ejecución

Para realizar la configuración del conector *mqtt-source*, se abre una nueva terminal y con ayuda del comando $curl$ se introducen las instrucciones que se muestran en la figura 4.27.

```
curl -s -X POST -H 'Content-Type: application/json'
http://132.248.32.20:8083/connectors -d '{
  "name" : "mqtt-source",
"config" : {
  "connector.class" : "io.confluent.connect.mqtt.MqttSourceConnector",
  "tasks.max" : "1",
  "mqtt.server.uri" : "tcp://132.248.32.20:1883",
  "mqtt.username" : "$User",
  "mqtt.password" : "*******",
  "mqtt.topics" : "test",
  "kafka.topic" : "testKafka",
  "key.converter":"org.apache.kafka.connect.storage.StringConverter",
  "value.converter":"org.apache.kafka.connect.converters.ByteArrayConverter",
  "key.converter.schemas.enable" : "false",
  "value.converter.schemas.enable" : "false",
  "confluent.topic.bootstrap.servers": "132.248.32.20:9092",
  "confluent.topic.replication.factor": "1",
  "confluent.license":""
  }
}'
```

Figura 4.27: Configuración de conector MQTT-Kafka

Posteriormente, es necesario reiniciar el proceso del conector distribuido. Para ello, se ejecuta la combinación de teclas Ctrl+C para deterner el proceso. Después, se ejecuta el comando: *./connect-distributed.sh ../config/connect-distributed.properties*, para ejecutar el proceso. El conector *mqtt-source* se configura para que escuche peticiones a través del puerto 8083. El proceso se monitorea para saber si está escuchando peticiones de red. Desde una terminal, se ingresa el comando: *curl -s "http://127.0.0.1:8083/connectors"*. La figura 4.28 muestra la salida de la respuesta mostrando el nombre del conector.

```
root@galileo:~# curl -s "http://132.248.32.20:8083/connectors"
["mqtt-source"]root@galileo:~#
```

Figura 4.28: Conector *mqtt-source* en ejecución

Para visualizar algunos detalles del conector MQTT-Kafka, se ejecuta la instrucción: *curl 127.0.0.1:8083*. La figura 4.29 muestra la salida del comando anterior.

```
root@galileo:~# curl 132.248.32.20:8083
{"version":"2.7.0","commit":"448719dc99a19793","kafka_cluster_id":"15u2RsW3T6-GhVpNGgerEQ"}root@galileo:~#
```

Figura 4.29: Detalles del conector *mqtt-source*

Para conocer el estatus del conector, se introduce el comando: *curl -s "http:// 127.0.0.1:8083/connectors/m source/status"*. La figura 4.30 muestra la salida del comando.

```
root@galileo:~# curl -s "http://132.248.32.20:8083/connectors/mqtt-source/status"
{"name":"mqtt-source","connector":{"state":"RUNNING","worker_id":"132.248.32.20:8083"},"tasks":[{"id":0,"state":"RUNNING","worker_id":"132.248
.32.20:8083"}],"type":"source"}root@galileo:~#
```

Figura 4.30: Estatus del conector *mqtt-source*

Por último, para listar los procesos que se están ejecutando en la máquina virtual de Java (JVM) se ejecuta el comando *jps*. La figura 4.31 muestra la salida del comando.

```
616025 ConnectDistributed
614650 QuorumPeerMain
616535 Jps
614839 Kafka
```

Figura 4.31: Procesos ejecutándose en la JVM

4.4. Integración de Kafka con Storm

Antes de iniciar el proceso de instalación de Apache Storm, es necesario contar con Python 2.6 o 2.7. debido a que la versión de Storm utilizada en este trabajo (2.2.0) no reconoce las versiones de Python 3 o superior. Para descargar el archivo de Storm se accede al sitio oficial de descargas de Apache Storm en https://storm.apache.org/downloads.html. Desde el sitio se busca el último lanzamiento, en este caso es, **apache-storm-2.2.0.tar.gz [PGP][SHA512]**. Posteriormente, se descarga el archivo comprimido desde el enlace proporcionado. Una vez descargado el archivo, se descomprime y se renombra la carpeta con

el nombre de *storm*. Posteriormente, se accede al directorio de configuración de Storm (*conf*) y se modifica el archivo *storm.yaml* con los parámetros que se muestran en la figura 4.32.

```
  GNU nano 4.8                    storm.yaml
########## These MUST be filled in for a storm configuration
storm.zookeeper.servers:
    - "132.248.32.20"
storm.local.dir: "/var/data/storm"
nimbus.seeds: ["132.248.32.20"]
supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703
ui.port: 8081
```

Figura 4.32: Archivo de configuración de Storm

Una vez modificados los parámetros de configuración, se crea el directorio de datos de Storm (*storm.local.dir*), tal y como se definió en el archivo de configuración. Para levantar el broker de Storm, primero, se inicia el servidor de ZooKeeper. Posteriormente, usando otra terminal, se accede al directorio bin de Storm y se lanza el nodo maestro (Nimbus) usando el comando: *storm nimbus*. La figura 4.33 muestra el despliegue del comando que levanta el proceso nimbus.

```
Running: /usr/lib/jvm/java-8-openjdk-amd64/bin/java -server -Ddaemon.name=nimbus -Dstorm.options= -Dstorm.home=/usr/local/storm -Dstorm.log.dir=/usr/local/storm/logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib:/usr/lib64 -Dstorm.conf.file= -cp /usr/local/storm/*:/usr/local/storm/lib/*:/usr/local/storm/extlib/*:/usr/local/storm/extlib-daemon/*:/usr/local/storm/conf -Xmx1024m -Djava.deserialization.disabled=true -Dlogfile.name=nimbus.log -Dlog4j.configurationFile=/usr/local/storm/log4j2/cluster.xml org.apache.storm.daemon.nimbus.Nimbus
```

Figura 4.33: Despliegue de Nimbus

En una terminal, desde el directorio bin de Storm se inicia el nodo esclavo (Supervisor) usando el comando: *storm supervisor*. La figura 4.34 muestra el despliegue del comando que levanta el proceso Supervisor.

```
Running: /usr/lib/jvm/java-8-openjdk-amd64/bin/java -server -Ddaemon.name=supervisor -Dstorm.options= -Dstorm.home=/usr/local/storm -Dstorm.log.dir=/usr/local/storm/logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib:/usr/lib64 -Dstorm.conf.file= -cp /usr/local/storm/*:/usr/local/storm/lib/*:/usr/local/storm/extlib/*:/usr/local/storm/extlib-daemon/*:/usr/local/storm/conf -Xmx256m -Djava.deserialization.disabled=true -Dlogfile.name=supervisor.log -Dlog4j.configurationFile=/usr/local/storm/log4j2/cluster.xml org.apache.storm.daemon.supervisor.Supervisor
```

Figura 4.34: Despliegue de Supervisor

En otra terminal, se accede al directorio bin de Storm para iniciar la interfaz de usuario (UI) usando el comando: *storm ui*. La figura 4.35 muestra el despliegue del comando que

levanta el proceso UI.

Figura 4.35: Despliegue de la UI de Storm

En un navegador web, se comprueba el despliegue de la UI de Storm. La UI está disponible ingresando la dirección IP del equipo y el puerto configurado para la UI (8081).La figura 4.36 muestra la interfaz gráfica de la UI.

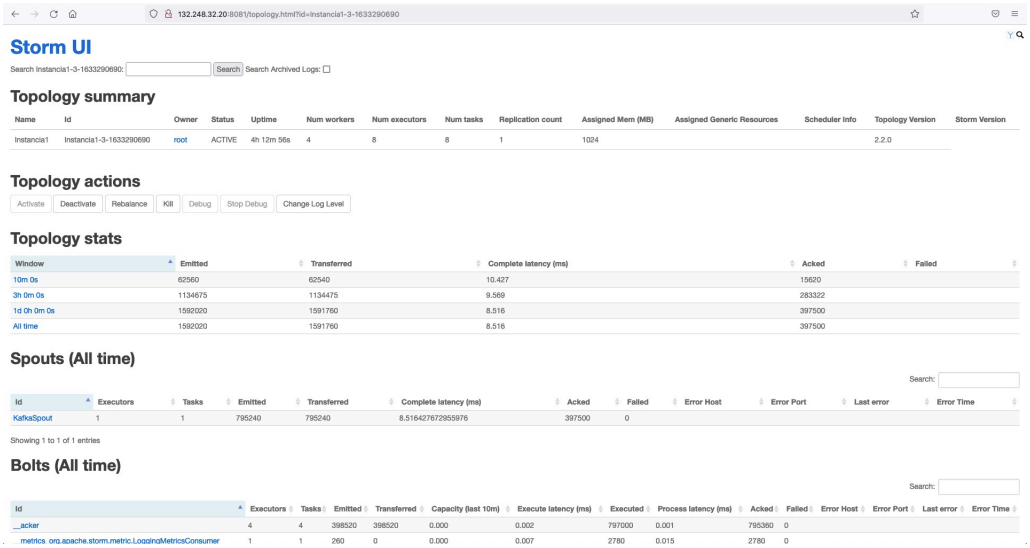


Figura 4.36: Interfaz de Usuario de Storm

Por último, se visualizan todos los procesos que se están ejecutando en la máquina virtual de Java (JVM) usando el comando *jps*. La figura 4.37 muestra el despliegue del comando que muestra que todos los procesos de Zookeeper, Kafka y Storm se están ejecutando.

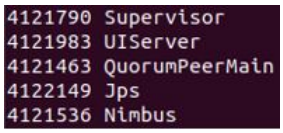


Figura 4.37: Procesos corriendo en la JVM

La arquitectura escalable distribuida de código abierto propuesta recopila, procesa y

almacena datos de ubicación de movilidad cada n segundos. Además, la configuración de Kafka y Storm garantiza tolerancia a fallas, escalabilidad y baja latencia. La siguiente sección analiza los resultados experimentales de la plataforma propuesta para tres casos de estudio.

Capítulo 5

VALIDACIÓN EXPERIMENTAL Y DISCUSIÓN

Esta sección describe la evaluación experimental de la plataforma propuesta y se describen las instancias del problema. Finalmente, los resultados de eficiencia computacional son reportados y discutidos.

5.1. Plataforma de desarrollo y ejecución

La topología Storm se implementó en el lenguaje de programación Java y la distribución de trabajos funciona de forma independiente según la distribución de la topología (ver figura 4.2). A cada componente spout/bolt se le asignó al menos un ejecutor (hilo) y a cada ejecutor se le asignó una tarea (instancia de spout/bolt). La evaluación experimental se realizó utilizando un procesador Intel i7-8700 (6 núcleos, 12 hilos) a 3,2 GHz, 32 GB de RAM y el sistema operativo Ubuntu 20.04 Linux.

5.2. Instancias de problemas y datos

A continuación se describen las instancias de problemas consideradas en cada caso de estudio. Las marcas de tiempo GPS del benchmark de trayectorias T-Drive [33] se utilizan para definir diferentes escenarios. Se definieron tres escenarios con 3500, 7500 y 10357 trayectorias GPS de 10357 rutas de taxi recopiladas del 2 al 8 de febrero de 2008, dentro

de Beijing. El formato de datos en un archivo contiene múltiples líneas con los siguientes campos, separados por una coma: idTaxi, fecha y hora, latitud, longitud.

El intervalo de tiempo entre dos marcas de tiempo de GPS consecutivas es de 10 minutos. Cada nombre de archivo en el conjunto de datos corresponde a la identificación del taxi y contiene la trayectoria del taxi. Los datos de la marca de tiempo GPS se envían desde el archivo al topic Mosquitto a través de un script $bash$ para realizar la evaluación experimental del procesamiento. La instancia del problema 1 incluye 3500 archivos de trayectoria. La instancia del problema 2 incluye 7500 archivos de trayectoria que contienen, y la instancia del problema 3 incluye los 10357 archivos de trayectorias.

5.3. Métricas de eficiencia computacional

La evaluación experimental se enfoca en evaluar el desempeño de la topología de la plataforma, compuesta por el $KafkaSpout$, el $SplitBolt$, el $CountBolt$, el $ReportBolt$ y el $MySQLBolt$ de persistencia de la base de datos (ver figura 4.2).

Se consideran dos métricas relevantes para la evaluación: *latencia* y *rendimiento* (throughput). La latencia evalúa el tiempo de comunicación (intra-worker) dentro de un proceso de trabajo asociado con un hilo ejecutor en la misma máquina/nodo de Storm. Además, cuando un servicio externo (broker de Kafka) interactúa con un componente de salida, la latencia evalúa el tiempo de comunicación entre topologías de la respuesta de un nodo Storm a un nodo Kafka a través de la red. Los valores de latencia más bajos significan un mejor tiempo de ejecución en general. A su vez, el rendimiento evalúa el número de tuplas procesadas con éxito, con respecto al número de marcas de tiempo GPS incluidas en cada instancia del problema. Un valor más alto de rendimiento representa una mejor eficiencia. Los valores contadores relacionados se recuperan de la interfaz de usuario de Storm y se utilizan para el rendimiento del throughput.

5.4. Resultados experimentales

Esta sección presenta y discute los resultados del análisis de eficiencia para los tres casos de estudio considerados, que involucran marcas de tiempo GPS de 10357 trayectorias de taxis en Beijing.

La Tabla 5.1 informa la cantidad de marcas de tiempo de GPS (*mtGPS*) en cada instancia del problema, la cantidad de tuplas procesadas con éxito (*acked*) por la topología, el rendimiento (*throughput*), la latencia promedio y el tiempo de actividad del sistema, es decir, el tiempo desde el comienzo de el experimento hasta que la topología deja de emitir tuplas. Los tiempos de latencia se informan en milisegundos.

Tabla 5.1: Contadores de eficiencia, rendimiento, latencia y tiempo de actividad de la topología para las tres instancias de problemas estudiados

instancia	*#mtGPS*	*acked*	*throughput*	*latencia (ms)*	*tiempo actividad*
#1	5742560	5112220	89.02 %	325.95	6h 8m
#2	12871223	12149800	94.39 %	332.12	13h 35m
#3	17662984	17490880	99.02 %	373.43	19h 4m

Los resultados de la Tabla 5.1 indican que la topología es capaz de procesar una cantidad significativamente grande de marcas de tiempo GPS en cada instancia en tiempos de ejecución razonables (considerando la plataforma computacional de gama baja utilizada). Se lograron valores de rendimiento entre 89 % y 99 %. No se logró el procesamiento de todo el conjunto de marcas de tiempo GPS, principalmente porque Storm usa tres colas internas en un nodo trabajador (worker node), lo que genera latencia adicional. La implementación desarrollada mostró buena escalabilidad, como lo demuestran los valores de latencia, los cuales no aumentaron de la instancia #1 a la #2, a pesar de que el número de GPS procesados aumentó en un factor de 2.25x. Además, la latencia solo aumentó un 14 %, cuando la dimensión del problema aumentó más de tres veces.

La Tabla 5.2 informa el número de tuplas transferidas por un componente bolt al siguiente componente en la topología (*transferred*), el rendimiento (*throughput*) y la latencia para cada componente bolt. Los componentes más relevantes son *SplitBolt* y *MySQLBolt* porque reciben datos directamente del spout. Por lo tanto, la cantidad de tuplas transferidas

en esos componentes bolt es igual o menor que la cantidad de marcas de tiempo de GPS almacenadas en el topic de Kafka para cada instancia. Los valores de rendimiento para el componente CountBolt no se informan porque el componente SplitBolt duplica el número de tuplas transferidas, como se informa en la columna transferida de la tabla. A su vez, los valores de rendimiento del componente ReportBolt no se notifican porque las tuplas que procesa y transfiere corresponden a valores de distancia, no a marcas de tiempo de GPS.

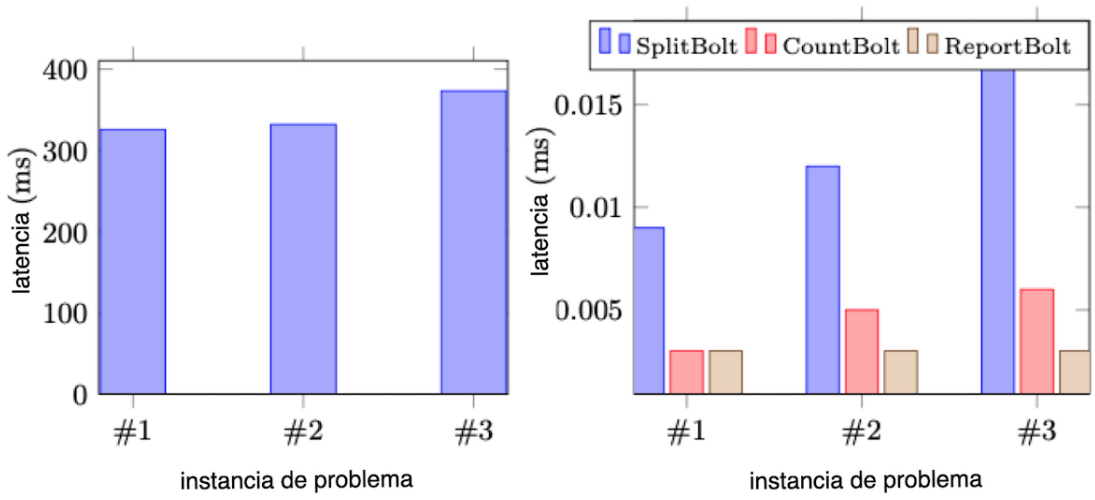
Tabla 5.2: Rendimiento y latencia asociados con componentes bolt

instancia	*bolt*	*transferred*	*throughput*	*latencia* (ms)
#1	SplitBolt	5402163	94.07 %	0.009
	CountBolt	11545907	–	0.003
	ReportBolt	5540620	–	0.003
	MySQLBolt	5363354	93.39 %	2.484
#2	SplitBolt	12343030	95.89 %	0.012
	CountBolt	25545907	–	0.005
	ReportBolt	12650720	–	0.003
	MySQLBolt	12303074	95.58 %	2.662
#3	SplitBolt	17345268	98.20 %	0.017
	CountBolt	35280140	–	0.006
	ReportBolt	17139500	–	0.003
	MySQLBolt	17147527	97.08 %	2.828

Los resultados de la Tabla 5.2 indican que los componentes bolt de interés (*SplitBolt* y *MySQLBolt*) pueden procesar una cantidad significativamente grande de marcas de tiempo para cada instancia en un tiempo de ejecución razonable, utilizando un ejecutor (hilo) para cada tarea asignada a cada componente bolt. Se lograron valores de rendimiento entre 93 % y 98 %. No se obtuvieron valores de rendimiento ideales debido a la pérdida de tuplas durante el procesamiento. La razón principal de esta pérdida es que el nivel de confiabilidad elegido para los componentes bolt no garantiza el procesamiento completo del mensaje; en su lugar, se centró en la eficiencia. Finalmente, el componente bolt que demandó mayor tiempo de ejecución fue el persistencia de datos (MySQLBolt), principalmente debido al tiempo de respuesta de la comunicación de la red con el servicio de base de datos externo.

La figura 5.1 resume gráficamente los resultados de latencia. Los valores de latencia

para la tupla completamente procesada a través de la topología se informan en la figura 3a. A su vez, la figura 3b compara los valores de latencia para cada componente bolt de la topología Storm aplicada.



(a) Latencia de topología (b) Latencias de los componentes bolt

Figura 5.1: Latencias de la topología Storm y componentes bolt

Los resultados en la figura 3a muestran claramente el comportamiento de latencia ligeramente creciente al resolver instancias de problemas con diferentes tamaños. A su vez, la figura 3b muestra que la latencia de los componentes bolt aumenta con el número de tuplas transferidas para las instancias del problema consideradas. *SplitBolt* aumenta la latencia principalmente debido a la tarea dividida de marcas de tiempo transferida al siguiente componente bolt. El componente *CountBolt* tiene un ligero incremento de latencia entre diferentes tamaños de instancias en relación con el número de tuplas transferidas desde SplitBolt, principalmente porque realiza un cálculo de CPU de baja demanda (es decir, cálculo de distancias). Finalmente, la latencia de *ReportBolt* tuvo el mismo valor en todas las instancias del problema.

En resumen, los resultados de la evaluación experimental demuestran que la topología Storm implementada en la arquitectura escalable distribuida de código abierto propuesta es

capaz de procesar una cantidad significativamente grande de marcas de tiempo GPS para cada instancia en tiempos de ejecución razonables. Además, el sistema mostró buenas propiedades de escalabilidad, ya que los valores de latencia aumentaron ligeramente al procesar una cantidad significativamente grande de marcas de tiempo GPS.

Capítulo 6

CONCLUSIONES Y TRABAJO FUTURO

6.1. Conclusiones

Este trabajo presentó arquitectura escalable distribuida de código abierto propuesta para el seguimiento de geolocalización en tiempo real en el contexto de las ciudades inteligentes, particularmente para aplicaciones STI.

La plataforma propuesta combina computación paralela y distribuida, y procesamiento de datos de geolocalización en tiempo real para análisis relevantes en ciudades inteligentes. Primero, la plataforma recopila marcas de tiempo de GPS de ubicación desde un dispositivo IoT sobre el protocolo MQTT al corredor Mosquitto; después de eso, el intermediario MQTT publica los mensajes en un tema de Apache Kafka mediante un conector Confluent. Luego, un consumidor de Kafka en particular inserta los datos en un spout de Storm (KafkaSpout) para que los componentes bolts los procesen en una topología paralela.

La validación experimental se centró en evaluar la eficiencia de la arquitectura propuesta a través de dos métricas relevantes de los componentes de Storm (latencia y rendimiento) utilizando marcas de tiempo de GPS reales de 10357 trayectorias de taxi desde Beijing que incluyen más de 17 millones de marcas de tiempo de GPS. Los principales resultados indican que la topología Storm propuesta es efectiva para procesar instancias de problemas que involucran una gran cantidad de marcas de tiempo GPS en tiempos de ejecución

razonables. Se lograron valores de rendimiento precisos, entre el 93 % y el 98 % para los componentes bolt y entre el 89 % y el 99 % para toda la topología. También se mostró un buen comportamiento de escalabilidad, ya que los valores de latencia no aumentaron significativamente al procesar más marcas de tiempo GPS.

Las principales líneas de trabajo futuro están orientadas a ampliar el sistema propuesto para computar otros indicadores y estadísticas esenciales para aplicaciones de STI, como son los sistemas de transporte público de México. Dichos análisis ciertamente serían valiosos para investigaciones relevantes como la sincronización de horarios de autobuses, el rediseño de la red de autobuses, los planes de movilidad sostenible y también el análisis del transporte privado.

Referencias

[1] Kumar, Sachin, Prayag Tiwari y Mikhail Zymbler: *Internet of Things is a revolutionary approach for future technology enhancement: a review*. Journal of Big Data, 6(1), 2019. 1

[2] Rodrigue, Jean Paul: *The Geography of Transport Systems*. Routledge, 5ª edición, 2020. 1, 10

[3] Alam, Muhammad, Joaquim Ferreira y José Fonseca: *Introduction to Intelligent Transportation Systems*. En Alam, Muhammad, Joaquim Ferreira y José Fonseca (editores): *Intelligent Transportation Systems*, capítulo 1, páginas 1–17. Springer International Publishing, 2016. 2, 9

[4] Hashem, Ibrahim Abaker Targio, Victor Chang, Nor Badrul Anuar, Kayode Adewole, Ibrar Yaqoob, Abdullah Gani, Ejaz Ahmed y Haruna Chiroma: *The role of big data in smart city*. International Journal of Information Management, 36(5):748–758, 2016. 2

[5] Li, Wenwen, Michael Batty y Michael F. Goodchild: *Real-time GIS for smart cities*. International Journal of Geographical Information Science, 34(2):311–324, 2020. 2, 10

[6] Sharma, Neha y Madhavi Shamkuwar: *Big Data Analysis in Cloud and Machine Learning*. En Mittal, Mamta, Valentina E. Balas, Lalit Mohan Goyal y Raghvendra Kumar (editores): *Big Data Processing Using Spark in Cloud*, capítulo 3, páginas 51–85. Springer, 2019. 2

[7] Hongyu, W., L. Jin, H. Zhenshan, F. Ruochen, M. Wenbo y H. Jian: *Research on parallelized real-time map matching algorithm for massive gps data*. Cluster Computing, 20:1123–1134, 2017. 3

[8] Ichinose, A., A. Takefusa, H. Nakada y M. Oguchi: *A study of a video analysis framework using kafka and spark streaming*. ISPRS International Journal of Geo-Information, página 2396–2401, 2017. 3

[9] Laska, Marius, Stefan Herle, Ralf Klamma y Jörg Blankenbach: *A Scalable Architecture for Real-Time Stream Processing of Spatiotemporal IoT Stream Data—Performance Analysis on the Example of Map Matching*. ISPRS International Journal of Geo-Information, 7(7), 2018. 4

[10] Galić, Z., E. Mešković y D. Osmanović: *Distributed processing of big mobility data as spatio-temporal data streams*. Geoinformatica, 21(2):263–291, 2017. 4

[11] Nesmachnow, Sergio, Sebastián Baña y Renzo Massobrio: *A distributed platform for big data analysis in smart cities: combining Intelligent Transportation Systems and socioeconomic data for Montevideo, Uruguay*. EAI Endorsed Transactions on Smart Cities, 2(5):320–347, 2017. 5, 11

[12] Nesmachnow, Sergio y Santiago Iturriaga: *Cluster-UY: Collaborative Scientific High Performance Computing in Uruguay*. Supercomputing. ISUM 2019. Communications in Computer and Information Science, 1151:188–202, 2019. 5

[13] Massobrio, Renzo y Sergio Nesmachnow: *Urban Mobility Data Analysis for Public Transportation Systems: A Case Study in Montevideo, Uruguay*. Applied Sciences, 10(16), 2020. 6

[14] Commission, European: *European Commission Mandate M/453 EN*, 2009. 10

[15] Batty, M., K.W. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis y Y. Portugali: *Smart cities of the future*. The European Physical Journal Special Topics, 214:481–518, 2012. 10

[16] Campos-Cordobés, Sergio, Javier del Ser, Ibai Laña, Ignacio (Iñaki) Olabarrieta, Javier Sánchez-Cubillo, Javier J. Sánchez-Medina y Ana I. Torre-Bastida: *Chapter 5 - Big Data in Road Transport and Mobility Research*. En Jiménez, Felipe (editor): *Intelligent Vehicles*, páginas 175–205. Butterworth-Heinemann, 2018. 11

[17] Jain, Ankit: *Mastering Apache Storm: Processing big data streams in real time*. Packt Publishing, Birmingham, UK, 2017, ISBN 978-1-78712-563-6. 11, 19, 20, 27

[18] Kumar, Manish y Chanchal Singh: *Building Data Streaming Applications with Apache Kafka*. Packt Publishing, Birmingham, UK, 2017, ISBN 978-1-78728-398-5. 11

[19] Marisol Barrón Bastida, Elizabeth de la Torre Romero, Alfredo Bueno Solano: *Análisis exploratorio sobre el uso de soluciones tecnológicas en las empresas de autotransporte de carga*. Informe técnico No. 531, Instituto Mexicano de Transporte, Querétaro MX, 2018. 13

[20] Llamas, Luis: *¿Qué es MQTT? Su importancia como protocolo IoT*. Disponible en https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/ (2019/04/17). 13, 14, 16, 17

[21] Llamas, Luis: *Qué son y cómo usar los Topics en MQTT correctamente*. Disponible en https://www.luisllamas.es/que-son-y-como-usar-los-topics-en-mqtt-correctamente/ (2019/06/19). 17

[22] Llamas, Luis: *Protocolos de comunicación para IoT*. Disponible en https://www.luisllamas.es/protocolos-de-comunicacion-para-iot/ (2020/09/12). 18, 19

[23] *Apache ZooKeeper*. http://zookeeper.apache.org. Revisado el: 2021-05-31. 19

[24] TutorialsPoint: *Apache Kafka - Cluster Architecture*. https://www.tutorialspoint.com/apache_kafka/apache_kafka_cluster_architecture.htm. Revisado el: 2021-05-31. 19, 23

[25] *Apache Kafka*. https://kafka.apache.org. Revisado el: 2021-05-31. 20

[26] Jonathan Leibiusky, Gabriel Eisbruch, Dario Simonassi: *Getting Started with Storm*. O'Reilly, United States of America, 2012. 26

[27] República, Instituto de Computación – Facultad de Ingeniería Universidad de la: *Persistencia en Sistemas O.O.* https://www.fing.edu.uy/tecnoinf/mvd/cursos/progapli/material/teo/PA-03-Persistencia.pdf. Revisado el: 2021-06-23. 30

[28] *MySQL*. https://www.mysql.com. Revisado el: 2021-05-31. 30

[29] Robledano, A.M.: *Qué es MySQL: Características y ventajas*. https://openwebinars.net/blog/que-es-mysql/. Revisado el: 2021-08-27. 30

[30] *MySQL :: MySQL 5.0 Reference Manual :: 19 Procedimientos almacenados y funciones*. http://download.nust.na/pub6/mysql/doc/refman/5.0/es/stored-procedures.html. Revisado el: 2021-05-31. 31

[31] Yuan, M.: *Conozca MQTT*. https://developer.ibm.com/es/articles/iot-mqtt-why-good-for-iot/. Revisado el: 2021-08-30. 37

[32] *Eclipse Mosquitto An open source MQTT broker*. https://mosquitto.org. Revisado el: 2021-08-30. 37

[33] Yuan, Jing, Yu Zheng, Xing Xie y Guangzhong Sun: *Driving with Knowledge from the Physical World*. En *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'11, página 316–324, New York, NY, USA, 2011. Association for Computing Machinery. 51