

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“EVALUACIÓN Y REDISEÑO DE UN MODELO DE PROCESAMIENTO DE STREAM ASISTIDO POR DISPOSITIVOS MÓVILES”

TEODORO FEDERICO SAAVEDRA OLIVA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Erika Rosas O.
Profesor Correferente: Diego Arroyuelo B.

Septiembre - 2019

RESUMEN

Resumen— Los sistemas de procesamiento de *stream* permiten procesar grandes cantidades de información en tiempo real. Estos utilizan un modelo basado en grafos, donde cada uno de sus nodos corresponde a un operador que realiza una tarea en específico. Dada la masividad del uso de dispositivos móviles, existe un modelo que busca integrar el poder de procesamiento de estos dispositivos a un sistema de procesamiento de *stream*, para lo que combina distintos componentes que se encargan de esta integración de forma que tenga en cuenta el balance de carga y la tolerancia a fallas.

En esta memoria se proponen cambios a este modelo para mejorar su desempeño, por lo que son implementados y evaluados bajo un ambiente simulado en el simulador de redes NS-3. Esta evaluación tiene como objetivo verificar el buen funcionamiento de los sistemas de balance de carga y tolerancia a fallas que se proponen, teniendo en cuenta la latencia de procesamiento, la utilización de los dispositivos móviles y el *overhead* del sistema.

Los resultados muestran que la integración de los dispositivos móviles libera carga del sistema de procesamiento de *stream*, sin tener un mayor impacto en la latencia de procesamiento y el modelo es capaz de recuperarse de las fallas que puedan suceder, con tiempos de recuperación de alrededor de 7.5 segundos y un promedio de eventos perdidos de 0.24 %.

Palabras Clave— Sistemas de procesamiento de *stream*, *Mobile Computing*, Sistemas distribuidos

ABSTRACT

Abstract— Stream processing systems allow to process large amounts of data in real time. These use a graph-based model, where each node corresponds to an operator who performs a specific task. Given the massive use of mobile devices, a model that seeks to integrate the processing power of mobile devices to a stream processing system, for which it incorporates different components that are responsible for this integration so that it takes into account the load balance and fault tolerance.

In this work, we present changes to this model to improve its performance, they are implemented and evaluated under a simulated environment in the network simulator NS-3. These evaluations aim to verify the proper functioning of the load balancing and fault tolerance systems, taking into account the processing latency, the processing utilization of the mobile devices and the system overhead.

The results show that the integration of the mobile devices release the load from the stream processing system, without having a greater impact on the processing latency and the model is able to recover from failures with a recovery time of around 7.5 seconds and an average of event losses of 0.24 %.

Keywords— Stream processing systems, Mobile Computing, Distributed Systems

GLOSARIO

Downstream: Corresponde a todo nodo que se encuentra en el mismo sentido que el flujo en un grafo de procesamiento con respecto a un nodo.

eps: Abreviación de Eventos Por Segundos.

Nimbus: Nodo maestro de un SPS que se encarga de la coordinación de este.

Operador *Source*: Operador encargado de recuperar desde alguna fuente y emitir los eventos hacia el SPS para que pueda ser procesado.

Operador de Procesamiento: Es el nodo de la topología de procesamiento que se encarga de realizar las tareas por las cuales pasara el flujo de eventos para se procesado.

Operador Terminal: Es el operador que se encarga de recibir el resultado final de la topología.

OM: *Operator Manager*, es el componente encargado de administrar y comunicar el estado de un nodo, principalmente móvil, para que estos no se sobrecarguen. Se encuentra junto con los *workers* en cada nodo.

RM: *Region Manager*, es el componente que esta encargado de administrar los recursos moviles con los que cuenta el sistema, teniendo la precaución de evitar fallas en el sistema. Se encuentra junto a *Nimbus* en el nodo maestro.

SPS: Sistemas de Procesamiento de *Stream*, por sus siglas en ingles *Stream Processing Systems*

Stream: Es el flujo de eventos que es generado por el operador *source* y que viaja a través del grafo de procesamiento, es representado por las aristas de este.

Upstream: Corresponde a todo nodo que se encuentre en sentido contrario al flujo en una grafo de procesamiento con respecto a un nodo.

Worker: Es el encargado de realizar la ejecución de las tareas dentro de un nodo.

ÍNDICE DE CONTENIDOS

RESUMEN	I
ABSTRACT	II
GLOSARIO	III
ÍNDICE DE FIGURAS	VII
ÍNDICE DE TABLAS	IX
ÍNDICE DE ALGORITMOS	X
CAPÍTULO 1: INTRODUCCIÓN	1
1.1 ANTECEDENTES, MOTIVACIÓN Y DESCRIPCIÓN DEL PROBLEMA	1
1.2 SOLUCIÓN PROPUESTA	2
1.3 OBJETIVOS	3
1.3.1 OBJETIVOS GENERALES	3
1.3.2 OBJETIVOS ESPECÍFICOS	3
1.4 METODOLOGÍA DE TRABAJO Y HERRAMIENTAS UTILIZADAS	3
1.4.1 METODOLOGÍA DE TRABAJO	3
1.4.2 HERRAMIENTAS UTILIZADAS	4
1.5 ALCANCES Y ORGANIZACIÓN DEL DOCUMENTO	4
1.5.1 ALCANCES	4
1.5.2 ORGANIZACIÓN DEL DOCUMENTO	4
CAPÍTULO 2: MARCO CONCEPTUAL	6
2.1 SISTEMAS DE PROCESAMIENTO DE STREAMING	6
2.1.1 ALGORITMOS DE BALANCE DE CARGA	7
2.1.2 TOLERANCIA A FALLAS	9
2.1.3 HERRAMIENTAS DE PROCESAMIENTO DE STREAM	11
2.1.3.1 Storm	11
2.1.3.1.1 Balance de carga con un acercamiento QoS-Aware en Storm	14
2.1.3.2 Spark	16
2.1.3.3 Flink	16
2.1.3.4 Simple Scalable Streaming System (S4)	17
2.2 MOBILE COMPUTING y FOG COMPUTING	17
2.2.1 MOBILE GRID COMPUTING	18
2.2.2 MOBILE CLOUD COMPUTING	18
2.2.3 FOG COMPUTING	19
2.2.4 PROCESAMIENTO DE STREAM SOBRE DISPOSITIVOS MÓVILES	19
2.2.4.1 MobileStorm	19
2.2.4.2 Symbiosis	20

2.3	ACTUAL MODELO DE PROCESAMIENTO DE STREAM ASISTIDO POR DISPOSITIVOS MÓVILES	20
2.3.1	ARQUITECTURA	20
2.3.2	FORMACIÓN DEL CLÚSTER Y MANTENIMIENTO DE LA REGIÓN	22
2.3.3	FLUJO DE EVENTOS Y BALANCE DE CARGA	24
2.3.4	TOLERANCIA A FALLAS	26
2.4	SIMULACIÓN DISCRETA POR EVENTOS Y NS-3	26
2.4.1	SIMULACIONES	26
2.4.2	SIMULACIÓN POR EVENTOS DISCRETOS	27
2.4.2.1	Principios de simulación	27
2.4.2.2	Definición de simulación discreta por eventos	28
2.4.2.3	Algoritmo para el avance de tiempo de la programación de eventos	29
2.4.3	ns-3 (Network Simulator version 3)	29
2.4.3.1	Flujo de trabajo	30
2.4.4	OTROS SIMULADORES	31
 CAPÍTULO 3: REDISEÑO DEL MODELO DE PROCESAMIENTO DE STREAM ASISTIDO POR DISPOSITIVOS MÓVILES		 32
3.1	MODELO DE TRANSMISIÓN DE EVENTOS	32
3.2	TOPOLOGÍA DE LOS OPERADORES	33
3.3	INGRESO NODOS MÓVILES AL SISTEMA	34
3.4	FORMACIÓN Y MANTENCIÓN DE CLÚSTERS	35
3.5	BALANCE DE CARGA HETEROGÉNEO	37
3.6	UMBRAL DE BALANCE DE CARGA	39
3.7	CONEXIONES ENTRE OPERATOR MANAGERS	40
 CAPÍTULO 4: DISEÑO EXPERIMENTAL		 42
4.1	EFFECTO DEL FLUJO DE EVENTOS SOBRE LA CARGA DE PROCESAMIENTO DEL SISTEMA	42
4.2	EFFECTOS DEL NÚMERO DE NODOS CONECTADOS SOBRE EL SISTEMA . . .	43
4.3	EVALUACIÓN DEL SISTEMA DE BALANCE DE CARGA	44
4.4	EVALUACIÓN DEL SISTEMA DE TOLERANCIA A FALLAS	45
 CAPÍTULO 5: RESULTADOS Y ANÁLISIS		 48
5.1	EFFECTO DEL FLUJO DE EVENTOS SOBRE LA CARGA DE PROCESAMIENTO DEL SISTEMA	48
5.2	EFFECTOS DEL NÚMERO DE NODOS CONECTADOS SOBRE EL SISTEMA . . .	50
5.3	EVALUACIÓN DEL SISTEMA DE BALANCE DE CARGA	51
5.3.1	EFFECTO DEL FLUJO DE EVENTOS SOBRE EL BALANCE DE CARGA	51
5.3.2	EFFECTOS DEL TIPO DE GRAFO DE PROCESAMIENTO SOBRE EL BALANCE DE CARGA	53
5.3.3	EFFECTOS DEL UMBRAL DE SOBRECARGA SOBRE EL BALANCE DE CARGA . .	54
5.3.4	EFFECTO DEL TIEMPO DEL CICLO MAPE SOBRE EL BALANCE DE CARGA . . .	56
5.3.5	ANÁLISIS FINAL PARA EL BALANCE DE CARGA	57

5.4	EVALUACIÓN DEL SISTEMA DE TOLERANCIA A FALLAS	58
5.4.1	EFFECTO DEL MODELO DE MOVILIDAD SOBRE LA TOLERANCIA A FALLAS . .	59
5.4.2	EFFECTO DEL TIEMPO DEL CICLO MAPE SOBRE LA TOLERANCIA A FALLAS . .	60
5.4.3	EFFECTO DEL FLUJO DE EVENTOS SOBRE LA TOLERANCIA A FALLAS	61
5.4.4	EFFECTO DEL TIPO DE GRAFO DE PROCESAMIENTO SOBRE LA TOLERANCIA A FALLAS	62
5.4.5	ANÁLISIS FINAL PARA LA TOLERANCIA A FALLAS	63
CONCLUSIONES		64
REFERENCIAS BIBLIOGRÁFICAS		67
A	ANEXO: PARÁMETROS DE SIMULACIONES	71
B	ANEXO: COMPLEMENTO DE RESULTADOS	73

ÍNDICE DE FIGURAS

2.1	Ejemplo de un grafo de procesamiento	7
2.2	Ejemplos de los distintos algoritmos de balance de carga	8
2.3	Ejemplo de una topología de <i>Storm</i>	12
2.4	Arquitectura de <i>Storm</i>	12
2.5	Estructura interna de un <i>worker</i> de <i>Storm</i>	13
2.6	Arquitectura de <i>Storm</i> con nuevos componentes agregados	15
2.7	Ejemplo del concepto de las distintas regiones de procesamiento	21
2.8	Arquitectura de <i>Storm</i> con la integración de los componentes del modelo	22
2.9	Modelo MAPE de dos niveles	23
2.10	Proceso de balance de carga entre nodos móvil y un nodo estático	25
2.11	Ejemplo del comportamiento del modelo cuando un nodo móvil falla	26
2.12	Durante la simulación el sistema solo cambia de estado en puntos discretos de tiempo t_i	28
2.13	Diagrama de flujo del algoritmo para el paso de tiempo de una simulación de eventos discretos.	29
3.14	Ejemplo de la transmisión de un evento en el modelo actual	32
3.15	Ejemplo de la transmisión de un evento en el modelo push	33
3.16	Ejemplo de una topología que cuenta con una bifurcación y su conexión entre los distintos operadores.	34
3.17	Ejemplo de una topología lineal.	34
3.18	Diagrama de Secuencia del ingreso de nodos al sistema en el modelo actual. . .	35
3.19	Diagrama de Secuencia del cambio al proceso de ingreso de nodos al sistema. . .	35
3.20	Representación gráfica del mapa virtual que posee el RM para la administración de recursos del sistema en el modelo actual.	36

3.21	Representación gráfica del mapa virtual que posee el RM para la administración de recursos del sistema en el modelo actual.	37
3.22	Ejemplo del proceso de liberación de carga de un nodo móvil sobrecargado. . . .	39
3.23	Ejemplo del proceso de regulación de la carga para un nodo móvil que se sobrecarga y luego se subcarga.	40
3.24	Diagrama de Secuencia del notificación entre nodos con el modelo actual.	40
3.25	Diagrama de Secuencia del notificación entre nodos a través del RM.	41
4.26	Topología lineal utilizada para realizar los distintos experimentos.	44
4.27	Topología bifurcada utilizada para realizar los distintos experimentos.	46
4.28	Topología trifurcada utilizada para realizar los distintos experimentos.	47
5.29	Resultados del experimento para observar los efectos del flujo de eventos sobre el sistema.	48
5.30	Resultados del experimento para observar los efectos del flujo de eventos sobre el sistema con solo nodo 3 nodos estáticos.	49
5.31	Resultados del experimento para observar los efectos del flujo de eventos sobre el sistema con solo nodo 3 nodos estáticos.	50
5.32	Gráficos de Utilización vs Tiempo de los resultados de los experimento para observar los efectos del flujo de eventos sobre el balance de carga.	52
5.33	Gráficos de Utilización vs Tiempo de los resultados de los experimento para observar los efectos del tipo de grafo de procesamiento sobre el balance de carga.	54
5.34	Gráficos de Utilización vs Tiempo de los resultados de los experimento para observar los efectos del umbral de sobrecarga sobre el balance de carga.	55
5.35	Gráficos de Utilización vs Tiempo de los resultados de los experimento para observar los efectos del tiempo del Ciclo MAPE sobre el balance de carga.	57
5.36	Gráficos de Latencia vs Tiempo de los resultados de los experimento para observar los efectos del tiempo del Ciclo MAPE sobre el balance de carga.	58
5.37	Diagramas del movimiento que presentan los nodos con los distintos modelos de movilidad.	60
A.1	Ajuste para simulación del consumo de energía en dispositivo móvil en NS-3. . . .	71

A.2	Ajuste para simulación del tamaño de eventos generados en NS-3.	72
B.1	Gráficos de Latencia vs Tiempo de los resultados de los experimento para observar los efectos del flujo de eventos sobre el balance de carga.	74
B.2	Gráficos de Latencia vs Tiempo de los resultados de los experimento para observar los efectos del tipo de grafo de procesamiento sobre el balance de carga.	75
B.3	Gráficos de Latencia vs Tiempo de los resultados de los experimento para observar los efectos del tipo de grafo de procesamiento sobre el balance de carga.	76

ÍNDICE DE TABLAS

5.1	Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del flujo de eventos sobre el balance de carga.	53
5.2	Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto de el umbral de sobrecarga sobre el balance de carga.	56
5.3	Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del tiempo del ciclo MAPE sobre el balance de carga.	56
5.4	Tabla resumen de los datos medidos en las pruebas para evaluar el efecto del modelo de movilidad sobre la tolerancia a fallas.	59
5.5	Tabla resumen de los datos medidos en las pruebas para evaluar el efecto del tiempo del ciclo MAPE sobre la tolerancia a fallas.	61
5.6	Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del flujo de eventos sobre el balance de carga.	62
5.7	Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del grafo de procesamiento de sobrecarga sobre el balance de carga.	62
A.1	Latencia de procesamiento utilizada para simular poder de cómputo de los nodo en NS-3	71
B.1	Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del flujo de eventos sobre la utilización de nodos.	73
B.2	Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del grafo de procesamiento sobre el sistema de balance de carga.	73
B.3	Tabla resumen de los datos medidos en las pruebas para evaluar el efecto del umbral de energía sobre la tolerancia a fallas.	76

B.4	Tabla resumen de los datos medidos en las pruebas para evaluar el efecto de la gama del dispositivo móvil sobre la tolerancia a fallas.	77
-----	---	----

ÍNDICE DE ALGORITMOS

2.1	Determinación de estado de recursos críticos.	23
2.2	Análisis de carga de nodos móviles.	25
3.3	Selección del Operador que se asociar a un nodo	38

CAPÍTULO 1

INTRODUCCIÓN

1.1. ANTECEDENTES, MOTIVACIÓN Y DESCRIPCIÓN DEL PROBLEMA

El concepto de Big Data [Mayer-Schönberger y Cukier, 2012], surge para atacar los conceptos de volumen, variedad y velocidad en la generación de los datos. Esta gran cantidad de datos se genera a través de Internet con tweets, fotos, vídeos y transacciones, entre muchos otros. Todos estos datos deben ser procesados por distintos motivos, por ejemplo, saber cuáles son los *Trending Topic* de Twitter, realizar análisis sobre las ventas de Amazon o simplemente saber el estado bancario de un usuario. Es decir, que existe la necesidad de generar información desde una gran cantidad de datos.

Para realizar esto es necesario almacenar, gestionar, procesar o analizar esta información y para lograrlo se necesita distribuir la ejecución de estas tareas en distintos nodos, de manera que se realicen de forma congruente, teniendo en cuenta la necesidad de escalabilidad que necesitan estos procesos. Existen acercamientos para lograr esto, los cuales no tienen en consideración el tiempo que se demorarán los datos desde que son generados hasta que son procesados. Uno de estos acercamientos son los sistemas de procesamiento en lote, en los cuales los datos se van almacenando para luego realizar procesamiento de todos estos, lo que conlleva un gran tiempo. Un ejemplo de este tipo de sistema es *Hadoop*¹ *MapReduce*, el cual utiliza distintos algoritmos para procesar esta gran cantidad de datos de forma distribuida en clúster, disminuyendo el tiempo que se demorarían si fueran procesados de forma secuencial.

Otro tipo de acercamiento que existe son aquellos que si tienen en consideración el tiempo en que se realiza este proceso, es decir, funcionan en tiempo real. Uno de estos, son los sistemas de procesamiento de *stream*, también conocidos como SPS por sus siglas en inglés *Stream Processing Systems* [Neumeyer et al., 2010a]. Estos son sistemas distribuidos desplegados en un clúster de computadores que se encargan de procesar los datos a medida que van arribando al sistema. Actualmente, una gran cantidad de empresas los utilizan, entre ellas Twitter, Yahoo!, Spotify, Alibaba, MercadoLibre, etc [Apache, 2018]. Este tipo de sistemas son los que se estudiarán en este documento.

Junto con esto, se ha observado la proliferación del uso de dispositivos móviles seguirá en el mismo curso como se ha visto en los últimos años, con un incremento de 5 mil millones de usuarios en 2017 a 5,9 mil millones en 2025 según GSMA [Intelligence, 2018]. Además, ha observado que son estos mismos dispositivos, que se encuentran en el borde de la red, los que generan la información a ser procesada. Y donde estos dispositivos poseen, a veces, igual capacidad de procesamiento que un servidor pequeño que se encuentra en la nube.

¹<http://hadoop.apache.org/>

Al paradigma de utilizar estos recursos de procesamiento, almacenamiento, etc., se le llama *Fog Computing* [Bonomi *et al.*, 2014] o computación en la niebla. Basando en esto, se ve la oportunidad de utilizar este tipo de paradigma para apoyar a los SPS que necesiten una alta capacidad de cómputo.

Utilizando la capacidad de procesamiento de los dispositivos móviles sería posible ayudar a los SPS, para lo cual es necesario un modelo que permita la administración de los recursos que poseen los distintos dispositivos móviles, teniendo en cuenta el balance de carga, la tolerancia a fallas, la latencia de procesamiento, etc., para que el sistema tenga un buen funcionamiento independiente de si que se presentan fallas en él. En [Morales, 2016] se establece este tipo de modelo, el cual se encarga de la petición y liberación de recursos respecto a la señal y la batería restante del dispositivo móvil. Para lograrlo se utiliza un modelo tipo MAPE [Kephart y Chess, 2003], que cuenta con 4 fases, Monitorear, Analizar, Planificar y Ejecutar, logrando así administrar los recursos y permitiéndole al modelo agregar el poder de procesamiento de distintos dispositivos móviles a un SPS.

Sabemos de acuerdo con lo estudiado en [Morales, 2016] que la hipótesis de que el agregar dispositivos móviles mejora la utilización del sistema de procesamiento de *stream*, pero se requiere evaluar la confiabilidad que posee el modelo para administrar los recursos de los dispositivos móviles, teniendo en cuenta la pérdida de información, los tiempos de recuperación en caso de falla y la administración de sus recursos limitados, como es la batería y su señal, así como también, la carga de procesamiento que se ejerce sobre estos dispositivos. Es necesario realizar la evaluación de cada uno de sus mecanismos de forma aislada, para determinar si es que el modelo administra estos recursos de forma eficiente y logra el objetivo de ayudar al desempeño de un SPS. Además, para poder ver si es que se necesita realizar modificaciones al modelo debido a fallas o mejoras en la eficacia de este.

1.2. SOLUCIÓN PROPUESTA

Respecto al actual modelo de procesamiento de *streams* asistido por dispositivos móviles presentado en [Morales, 2016], se busca identificar posibles fallas y/o zonas en las cuales se puede mejorar el desempeño del actual modelo, teniendo en cuenta la pérdida de información, los tiempos de recuperación en caso de falla y los recursos limitados con que cuentan los dispositivos móviles. Junto con esto, se presentarán soluciones o mejoras frente lo que se ha identificado, dando los fundamentos correspondientes para dichas decisiones.

Dada la necesidad de evaluar la confiabilidad del modelo de procesamiento de *stream* asistido por dispositivos móviles, es necesario confeccionar distintos experimentos, evaluando cada una de las aristas del modelo presentado, es decir, evaluar el modelo en su capacidad de balancear carga, tolerancia a fallas, administración de batería, latencia de procesamiento, asignación de tareas y/o recursos, entre otros. El objetivo es investigar la capacidad de control que posee el modelo sobre los recursos administrados.

1.3. OBJETIVOS

1.3.1. OBJETIVOS GENERALES

Rediseñar y evaluar el rendimiento, tolerancia a fallas y *overhead* de un modelo de un sistema de procesamiento de *stream* asistido por dispositivos móviles.

1.3.2. OBJETIVOS ESPECÍFICOS

1. Identificar falencias y/o oportunidades de mejora en el sistema actual de procesamiento de *streams* asistido por dispositivos móviles y proponer alternativas de solución.
2. Implementar el nuevo sistema modificado, de manera modular en el simulador NS-3.
3. Evaluar el rendimiento, tolerancia a fallas y *overhead* del nuevo modelo del sistema de procesamiento de *stream* asistido por dispositivos móviles.

1.4. METODOLOGÍA DE TRABAJO Y HERRAMIENTAS UTILIZADAS

1.4.1. METODOLOGÍA DE TRABAJO

Dada las características y objetivos planteados para el desarrollo de esta memoria, se utilizará como base el método descrito en [Hernández Sampieri *et al.*, 2006] de procesos de la investigación mixta, junto con el método experimental para simulación de algoritmos establecido en [McGeoch, 1996] para el diseño, desarrollo y ejecución de los experimentos a realizar. Las distintas etapas se detallan a continuación:

- **Elaboración de un marco teórico:** se presentan distintos tipos de SPS, estableciendo las distintas técnicas utilizadas por cada uno de estos, teniendo en cuenta principalmente la tolerancia a fallas y los métodos de planificación de tareas. Además, se presentan distintos trabajos que exponen el funcionamiento de las redes de dispositivos móviles para la ejecución de calculo computacional.
- **Rediseño del modelo:** respecto a lo estudiado en el marco teórico, se establece cuales son los cambios necesarios para poder atacar las debilidades identificadas, para luego implementar los cambios establecidos, de forma de poder mejorar el rendimiento general del modelo.
- **Diseño de las evaluaciones:** se establecen distintas evaluaciones al modelo en estudio, teniendo en cuenta las diferentes aristas correspondientes al funcionamiento de este, utilizando un proceso iterativo e incremental para el desarrollo de estas pruebas.

- **Análisis de resultados:** se obtienen métricas con el objetivo de poder establecer concretamente cuales son las fortalezas y debilidad que el modelo posee, haciendo distinción de cuales son las que se pueden manipular.

1.4.2. HERRAMIENTAS UTILIZADAS

Para el desarrollo de esta memoria se requiere simular redes de dispositivos móviles y distintos nodos computacionales, para esto se utilizó el software ns-3², que es un simulador de eventos discretos que utiliza el lenguaje de programación C++ para el desarrollo de sus componentes y las distintas interfaces de simulación.

Para poder ejecutar estas simulaciones se utilizaron 2 servidores con dos procesadores Xeon E5-2680 v3 con 12 cores *hyperthreading* cada uno, contando uno con 128 GB de RAM y el otro con 64 GB de RAM.

1.5. ALCANCES Y ORGANIZACIÓN DEL DOCUMENTO

1.5.1. ALCANCES

- Las simulaciones están basadas en un *benchmarking* realizado en [Morales, 2016]. En esta experimentación se mantendrán los valores obtenidos en ese trabajo.
- Los problemas de seguridad respecto a nodos maliciosos quedan fuera del marco de esta investigación.

1.5.2. ORGANIZACIÓN DEL DOCUMENTO

Este documento, continua con el desarrollo pertinente para lograr los objetivos presentados en este capitulo, lo que se presenta en distintos capítulos los cuales siguen la metodología presentada dentro de este capítulo.

El Capítulo 2 presenta un marco teórico que sirve como base para el entendimiento de los conceptos que se investigan en este documento, donde se incluye los tópicos más importantes que incumben con el modelo de procesamiento de *streams* asistido por dispositivos móviles y la presentación de dicho diseño modelo.

El Capítulo 3 presenta el rediseño del modelo con los cambios sugeridos para el mejoramiento del desempeño de este, junto con las justificaciones necesarias para dichos cambios.

²<https://www.nsnam.org/>

El Capítulo 4 presenta las evaluaciones a realizar sobre el sistema, cuales son las variables que afecta a cada uno de ellos y los valores que toman dentro de estas mismas, además, se mencionará si es que existen algunos resultado esperado para estas evaluaciones.

El Capítulo 5 presenta los resultados obtenidos de las evaluaciones y el análisis de estos. Donde en estos análisis se realizarán sobre los efectos de flujo de eventos sobre la carga del sistema, los efectos del número de nodos sobre el sistema, el sistema de balance de carga y el sistema de tolerancia a fallas.

Finalmente, el Capítulo 6 presenta las conclusiones obtenidas a través de este documento, junto con cual será el trabajo futuro de esta investigación. Además, se presentan referencias bibliográficas y los anexos necesarios para el desarrollo de este documento.

CAPÍTULO 2

MARCO CONCEPTUAL

Para comprender los conceptos necesarios para este trabajo se desarrolla un marco conceptual. Este permite unir los distintos tópicos a tratar durante este documento permitiendo establecer una base para los conocimientos necesarios para comprender la solución propuesta. De esta forma este capítulo se encargará de brindar los conceptos necesarios para comprender de donde viene ciertas características de los sistemas de procesamiento de *streaming*, cuales son las distintas configuraciones de dichas características entre otros. Además, se presentarán trabajos en la literatura que sean relevantes para este documento, de los cuales se tomarán conceptos e ideas para el mejoramiento del modelo a presentar.

La estructura de este capítulo es la siguiente: primero se expondrán que son los sistemas de procesamiento de *streaming* en profundidad. Segundo se expondrá que es *fog computing* y *mobile assisted computation*. Tercero se presentará el actual modelo de procesamiento de *stream* asistido por dispositivos móviles. Y finalmente, que son las simulaciones de eventos discretos y ns-3.

2.1. SISTEMAS DE PROCESAMIENTO DE STREAMING

Los sistemas de procesamiento de *stream*, o *stream processing system* (SPS), son un tipo de sistemas distribuidos, los cuales se utilizan para procesar un gran flujo o *stream* de información en tiempo real. Este flujo se compone por eventos o tuplas [Arasu *et al.*, 2003], donde cada evento puede provenir de distintas fuentes de información. Por ejemplo, *tweets* enviados durante un evento en vivo o los registros generados por un sistema transaccional.

Los SPS son sistemas escalables y normalmente capaces de adaptarse a los cambios en los flujos de información generados por las diferentes fuentes y así poder dar soporte al poder de computo necesario para procesar dicho flujo [Cherniack *et al.*, 2003]. Otra característica que poseen es su capacidad de ser tolerante a fallas, permitiendo al sistema poseer la robustez necesaria para minimizar la pérdida de información en casos de fallo.

Las aplicaciones de SPSs están representadas por grafos acíclicos dirigidos (DAG) llamados grafos de procesamiento. Este grafo se encarga de la lógica del procesamiento de los eventos, donde cada nodo representa un operador que se encarga de procesar dichos eventos y los vértices representan el sentido del flujo de estos eventos. Siendo esta representación desplegada en un clúster de nodos físicos utilizando recursos virtualizados, permitiendo asignar estos recursos de forma eficiente a los operadores, ya sea de forma atómica o replicada [Balazinska *et al.*, 2005].

Los operadores pueden realizar tareas de diversa complejidad como unión, filtrado,

conteo, entre otros. Además, a los operadores anteriores en el flujo de los datos se denominan *upstream*, mientras que para el caso contrario se denominan *downstream* [Toshniwal et al., 2014]. Se puede observar un ejemplo de un grafo de procesamiento en la Figura 2.1, el cual procesa eventos de registros de un sistema.

En el ejemplo se puede ver que el nodo *indexer* se encuentra en el *downstream* respecto a el nodo *rule-based filter* y este a su vez se encuentra en el *upstream* respecto al otro. Además, se observa que los vértices representan el flujo y la dirección en que se envían los eventos. Finalmente, todos los grafos de procesamiento deben poseer al menos un nodo fuente, o *source*, y al menos un nodo consumidor, o *consumer*, como se ve en la Figura 2.1.

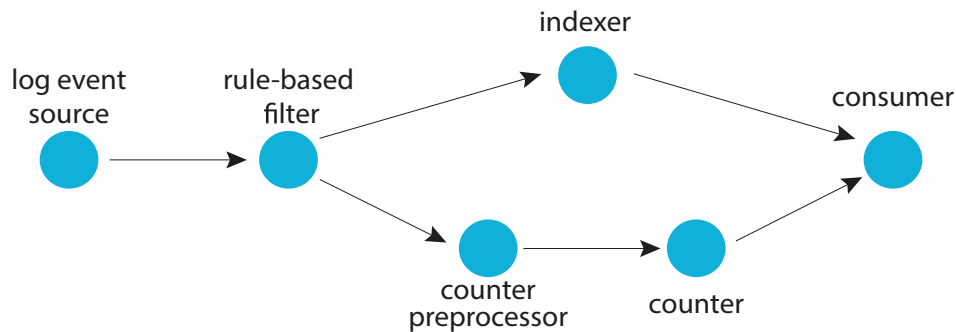


Figura 2.1: Ejemplo de un grafo de procesamiento
Fuente: [Cardellini et al., 2015].

2.1.1. ALGORITMOS DE BALANCE DE CARGA

Los sistemas distribuidos cuentan con algoritmos de planificación o balance de carga para gestionar los recursos del sistema, que buscan mejorar el desempeño del sistema a través de la distribución de la carga de trabajo entre los distintos nodos que componen el sistema. Para esto se analizan distintos indicadores como el *throughput*, que refleja el tiempo de respuesta promedio de las tareas asignadas. La cola de recursos y la cola de tareas asignadas a la CPU del nodo son otros indicadores, los cuales tienen relación con el tiempo de respuesta a la asignación de una tarea a un nodo.

Hay distintos tipos de algoritmos de planificación, estos se categorizan según el tiempo en que se realiza el balance, siendo preventivo o reactivo [R. Kumar Singh y Saxena, 2012], o según la forma en que balancea la carga de los operadores [Casavant y Kuhl, 1988], estos son clasificados en:

- **Algoritmos estáticos:** las tareas son asignadas de manera fija a cada operador sin considerar factores externos. Este tipo de algoritmo es utilizado por su simplicidad, pero no toma en cuenta la heterogeneidad en las tareas por lo que no balancea de manera eficiente en ambientes con tareas de distinto tamaño.

- **Algoritmos dinámicos:** captando información del sistema realizan decisiones en la planificación de las tareas. Como lo es el algoritmo de *Weighted-round-robin* [Katevenis *et al.*, 1991], que asigna un peso a cada nodo y que luego, se le asignarán las tareas considerando dicho peso.
- **Algoritmos adaptativos:** estos son algoritmos de tipo dinámicos, pero estos cambian sus políticas de planificación en base al estado del sistema. De esta manera permite disminuir la latencia y/o la pérdida de paquetes en condiciones de mayor congestión. Como es el caso de *load-aware weighted round robin* [I. Saidu *et al.*, 2014], donde el peso asignado a cada nodo cambia durante el tiempo.

En la Figura 2.2 se puede observar ejemplos de los tres tipos de algoritmos. En la Figura 2.2.a se observa el comportamiento de un algoritmo estático donde el planificador asigna las tareas TX de forma secuencial a los nodos, en la Figura 2.2.b el planificador dependiendo de los pesos de cada nodo, en este caso asigna dos tareas a el nodo 1 para luego asignar tres tareas al nodo 2 y así sucesivamente. Finalmente, en la Figura 2.2.c el planificador asigna tareas de la misma forma que en el caso anterior pero los pesos de los nodos cambia en el periodo $i+1$ respecto a un periodo i .

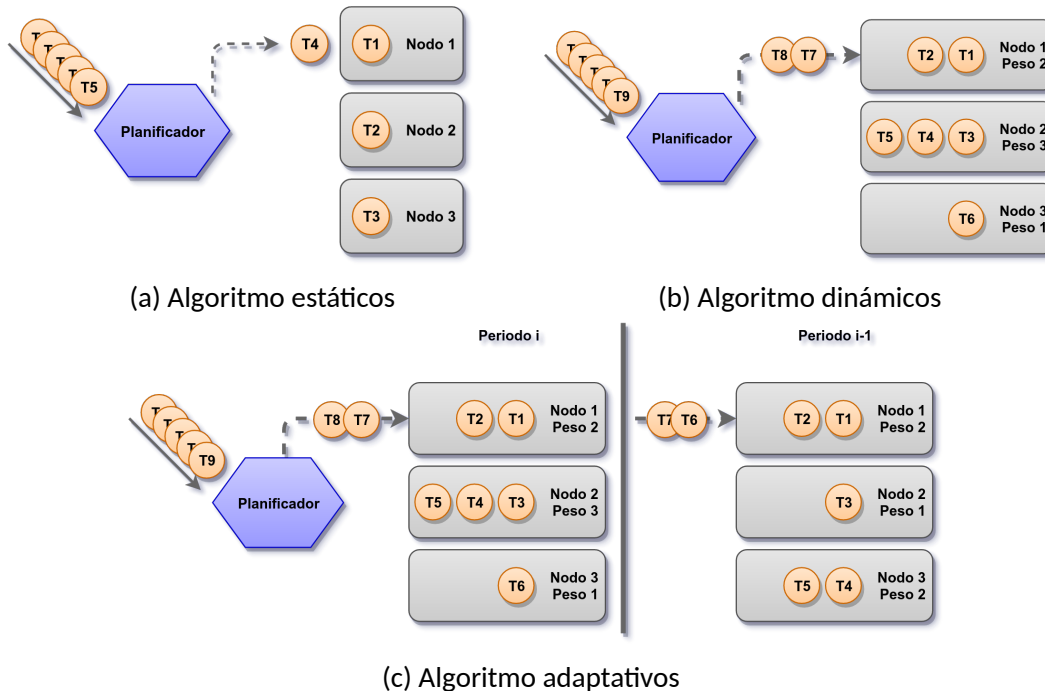


Figura 2.2: Ejemplos de los distintos algoritmos de balance de carga
Fuente: [Morales, 2016].

Otros tipos de enfoques más complejos utilizados en los SPS son, por ejemplo, *Delay Scheduling* [Zaharia *et al.*, 2010], el cual es utilizado por *Spark* y se centra en el retraso del procesamiento de los eventos, es decir, si un evento ha sido retrasado por una cierta cantidad de tiempo este es reenviado a otro operador para ser procesado. En [Rychlý *et al.*, 2015] se

presenta otro enfoque, el cual tiene en consideración la heterogeneidad del clúster, en este enfoque se debe considerar los recursos de hardware con que se cuentan y las características de los operadores que se piensan implementar, realizando un *benchmark* que evalúe el desempeño de los operadores en las máquinas que poseen distintos recursos, asignando cada operador a la máquina que le permita tener mejor desempeño, buscando así el mejor desempeño para la aplicación en general.

En *Storm* [Toshniwal et al., 2014], la distribución de eventos puede ser configurada de ocho formas distintas, entre esas, el algoritmo llamado *Shuffle grouping* distribuye los eventos de forma aleatoria entre los operadores, por lo que no balanceara la carga de manera correcta si es que los operadores no son homogéneos, debido a que no todos los eventos generan la misma carga en el sistema, y como ya se ha mencionado, los eventos pueden ser procesados de mejor forma por algún operador que posea características más *ad-hoc* a él. Otro de estos algoritmos es *Field grouping*, el cual distribuye los eventos según sus atributos, causando que los eventos con los mismos atributos sean procesados por el mismo operador.

En [Cardellini et al., 2015] se presenta un sistema de balance de carga con un enfoque de tipo QoS-Aware (Consiente de la Calidad de Servicio), el cual permite al sistema cambiar la posición de los operadores de forma automática para que puedan estar lo más cerca posible a la fuente de los datos y al consumidor final de estos, para esto agrega nuevos componentes al sistema, lo que mejora la latencia del sistema, descongestiona el tráfico entre los nodos del sistema y mejora la respuesta del sistema a aumentos bruscos en la utilización de los nodos, permitiendo una estabilización del sistema de forma más efectiva.

2.1.2. TOLERANCIA A FALLAS

Es necesario que un sistema distribuido sea robusto debido a la susceptibilidad que tienen estos sistemas frente a los fallos, debido a que algún nodo del sistema sea interrumpido o deje de funcionar por algún error. A diferencia de los sistemas de procesamiento de información por lotes, en los SPS el tiempo de recuperación de fallas es un aspecto fundamental en su desarrollo, esto se debe a la influencia que tiene en el tiempo de procesamiento de un evento para los resultados que entrega el sistema. Si es que llegara a fallar un nodo en el clúster donde el SPS es ejecutado, es necesario que el sistema sea capaz de seguir realizando las tareas que dicho nodo soportaba. Además, el sistema debe ser capaz de reasignar las tareas de los nodos caídos manteniendo un estado integro de la información que el nodo posee y así obtener los resultados con la precisión necesaria. El sistema debe asegurar que la pérdida de información sea mínima.

Según la integridad de la información después de una recuperación es como se clasifican los algoritmos de tolerancia a fallas [Hwang et al., 2005]:

- **Precise recovery:** no existe pérdida de información por lo que presenta una alta latencia de recuperación, además utiliza una gran cantidad de recursos debido a que

almacena toda la información que es procesada en el SPS para entregar una respuesta como si no hubiera ocurrido el fallo para el usuario.

- **Rollback recovery:** para este tipo algoritmo los operadores deben almacenar sus eventos de salida, por lo que en caso de falla de un operador *downstream* se podrán re-entregar los eventos que se perdieron a un nuevo operador saludable, es decir, que se encuentre sin problemas. Esto evita la pérdida de información, pero no garantiza la precisión de esta, pudiendo obtener una respuesta similar a la cual se hubiera obtenido si es que no hubiera sucedido alguna falla. Esta pérdida de precisión se debe a que, al ocurrir una falla, el operador, que se encuentra en estado saludable, puede volver a procesar eventos que ya habían sido procesado antes de la falla, lo que llevaría a un resultado distinto.
- **Gap recovery:** también conocido como amnesia, es el tipo de algoritmo que cuenta con la mayor velocidad de recuperación, debido a que solo consiste en levantar un nuevo operador, lo que hace que toda la información se pierda definitivamente y el nuevo operador empiece a procesar eventos sin el estado anterior. Este tipo de algoritmo es utilizado por S4 como *Load Shedding* [Neumeyer *et al.*, 2010a].

Dentro de los SPS existen dos modos de tolerancia a fallas, pasivo o activo, El modo pasivo implica que los operadores guarden el estado de las tareas en ejecución y al momento de ocurrir una falla el sistema crea un nuevo operador con el estado almacenado. En cambio, el modo activo utiliza replicación de las tareas en otros operadores, así reduciendo los tiempos de recuperación del sistema, lo que significa una mayor utilización de recursos por parte del sistema. Existen también acercamiento híbridos, donde se almacena tanto el estado de la tarea, de forma periódica, como también existe la replicación de estas dependiendo de los recursos con que vaya contando el sistema, lo que permite que al momento de fallar un operador primero se busque una réplica y si es que no existe se pueda levantar un nuevo operador con el estado almacenado [Su y Zhou, 2015] [Martin *et al.*, 2015].

Una parte central de los mecanismos de tolerancia a fallas es el *checkpointing*, el cual captura el estado del sistema y sus operadores de forma periódica permitiéndole al sistema volver a estos puntos en el estado del sistema en caso de que este falle. Cada SPS tiene formas distintas de realizar este proceso dependiendo del tipo de semántica de entrega que el sistema asegura, estas semánticas son [Akka, 2019]:

- **at-most-once:** En esta semántica cada evento que es entregado al sistema va a ser enviado a los operadores solo una vez, es decir, va a ser procesado por los operadores de procesamiento o va a ser consumido por el operador consumidor a lo más una vez. Básicamente, no hay garantías de que el mensaje no se vaya a perder.
- **at-least-once:** En esta semántica cada evento que es entregado al sistema se intentará enviar a los operadores varias veces hasta que al menos una vez sea entregado exitosamente y de forma completa al operador consumidor. Es decir, el mensaje puede ser recibido duplicado, pero nunca se va a perder.

- **exactly-once:** En esta semántica cada evento que es entregado al sistema va a ser entregado a los operadores siempre de forma exitosa hasta llegar al nodo consumidor, garantizando que el mensaje nunca se va a perder o recibir duplicado.

2.1.3. HERRAMIENTAS DE PROCESAMIENTO DE STREAM

En esta sección se presentarán alguna de las herramientas SPS que son usadas en la actualidad, destacando principalmente a *Storm*[Storm, 2018], la cual es en la que está basada el modelo que se presenta en este documento por lo que se realiza un análisis más profundo de su arquitectura y funcionamiento. Además, se presentan otras herramientas con mayor presencia en la actualidad como lo es *Spark*[Spark, 2016].

2.1.3.1. Storm

Apache Storm[Storm, 2018] es un sistema distribuido de procesamiento en tiempo real, el cual permite procesar flujos de datos, utilizando un modelo de programación simple. *Storm* es tolerante a fallas, asegurando que cada evento sea procesado bajo la semántica de *exactly-once*. Empresas como *Spotify*, *Yahoo!*, *Alibaba*, *Twitter*, entre otros, utilizan *Storm* en sus operaciones del día a día³.

Storm utiliza un grafo de procesamiento donde existen dos tipos de nodos, *spouts* y *bolts*, que son conectados a través de *streams*, los cuales representan una secuencia ilimitada de tuplas que contienen la información que será procesada por el sistema. Un *spout* es un operador que se encarga de leer tuplas de fuentes externas y emitir estas tuplas hacia la topología, estos pueden ser confiables o no confiable, donde un *spout* confiable es capaz de volver a generar una tupla que no fue procesada completamente por *Storm* y en caso contrario un *spout* no confiable olvida las tuplas apenas las emite.

En los *bolts* es donde se realiza el procesamiento, estos operadores pueden hacer desde *filtering*, *joins*, *aggregations*, ejecución de funciones, realizar llamadas a una base de datos y más. Los *bolts* realizan tareas simples, por lo que para realizar tareas más complejas es necesario realizarlas múltiples pasos, por tanto, es necesario múltiples *bolts*. En la Figura 2.3 se puede ver un ejemplo de una topología de *Storm*, donde las llaves representan a los *spouts* y los rayos representan a los *bolts*.

Storm cuenta con tres componentes principales: *Zookeeper*, *Nimbus* y los nodos trabajadores o *workers*. *Zookeeper* es la capa de coordinación de aplicaciones que trabajan de forma distribuida y concurrente. Esta se encuentra distribuida entre los *workers* donde funciona como un sistema de archivos, en donde cada uno de estos representa a un *worker*, permitiendo así una jerarquía dentro de los mismos. Esta arquitectura le permite garantizar consistencia

³<http://storm.apache.org/releases/current/Powered-By.html>

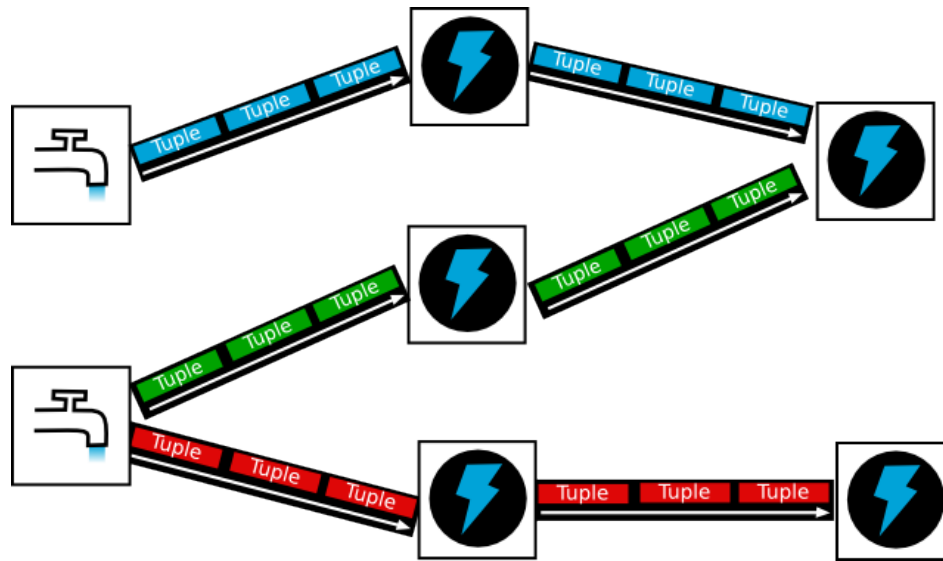


Figura 2.3: Ejemplo de una topología de Storm
Fuente: [Storm, 2018].

secuencial y atomicidad, además de proporcionar al usuario con una vista única del sistema y *timeliness*, es decir, garantiza que la vista del sistema este actualizada cada cierta cantidad de tiempo (del orden de las décimas de segundo).

En Storm, *Zookeeper* se encarga de transmitir información y el estado del sistema entre *Nimbus* y los *workers*, siendo *Nimbus* el nodo maestro de Storm, encargándose de distribuir las tareas a los *workers* y ejecutar políticas que permitan corregir problemas en el sistema en caso de falla. Finalmente, los *workers* son administrados y levantados por procesos supervisores. En la Figura 2.4 se muestra un diagrama de la arquitectura que posee Storm donde se observa claramente la jerarquía entre los componentes.

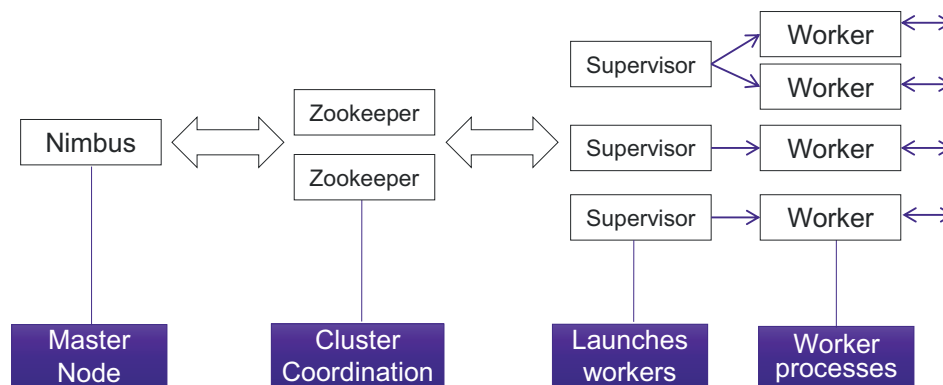


Figura 2.4: Arquitectura de Storm
Fuente: [Peng, 2016].

Los *workers* son procesos Java que ejecutan parte de una topología en específico, donde cada *worker* puede correr uno o más *executors*. Estos *executors* son *threads* que ejecutan tareas

pertenecientes a un nodo del grafo de procesamiento, ya sea un *bolt* o un *spout*, donde finalmente son estas tareas las encargadas de realizar el procesamiento del sistema, como también, permite la replicación de estas para evitar sobrecarga. Además, cada *worker* posee una cola de recepción de tupla y otra cola de salida. Estas colas y los demás componentes de un *worker* se pueden ver en la Figura 2.5, la cual muestra la relación entre los distintos componentes de un *worker*. Las tuplas que llegan al *worker* se envían a la cola de entrada que posee el *executor* asociado, el cual toma la tupla y ejecuta las tareas definidas por los operadores. Cada operador puede ejecutar cuantas tareas se desee a través del clúster, pero el número de tareas por un operador siempre se mantiene constante durante la vida de la topología, pero el número de *executors* por cada operador puede cambiar, es decir, que como mínimo *Storm* correrá una tarea por *executor*.

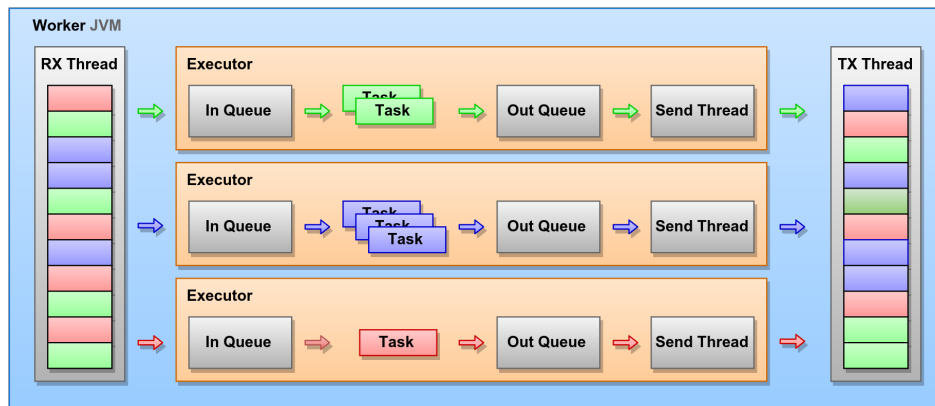


Figura 2.5: Estructura interna de un *worker* de *Storm*

Fuente: [Morales, 2016].

Por otro parte, es posible cambiar la manera en que los envíos de tuplas a las tareas réplicas de un *bolt* es planificada, conocido como *stream grouping*. Las distintas maneras en que se pueden planificar son:

- **Shuffle grouping:** las tuplas son distribuidas de forma aleatoria hacia las tareas del *bolt* de forma que se garantice que cada réplica del *bolt* recibirá una cantidad relativamente igual de tuplas.
- **Fields grouping:** los tuplas son distribuidas por un atributo de las tuplas, por lo que las tuplas que tengan el mismo valor en dicho atributo irán siempre a la misma tarea.
- **Partial Key grouping:** es similar a *Fields grouping* pero aplica un balance de la carga entre dos *bolts* del *downstream*, probando ser más eficiente con los recursos cuando los datos tiene un sesgo [Nasir et al., 2015].
- **All grouping:** el *stream* es replicado para todas las tareas del *bolt*.
- **Global grouping:** todo el *stream* va específicamente a la tarea del *bolt* cuyo id sea el menor.

- **None grouping:** no importa como el *stream* es distribuido. Actualmente, equivale a *Shuffle grouping* pero eventualmente *Storm* obligará a estos *bolts* a ser ejecutar en el mismo *thread* del *bolt* o *spout* de donde están suscritos (si es que es posible).
- **Direct grouping:** el productor de la tupla decide que tarea del consumidor la recibe.
- **Local of shuffle grouping:** Si el *bolt* posee una o más tareas en el mismo *worker*, las tuplas serán distribuidas a solo esas tareas en progreso. De otra manera, se realizará *Shuffle grouping*.

2.1.3.1.1. Balance de carga con un acercamiento QoS-Aware en *Storm*

Como se mencionó anteriormente, existe un algoritmo balanceador de carga que toma en cuenta la calidad de servicio (QoS) de la red de nodos distribuidos para poder automatizar la locación de los operadores de forma que estos se encuentren lo más cercano posible a las fuentes de datos o como también a los consumidores finales. Para esto, los autores en [Cardellini *et al.*, 2015] agregan un organizador de carga, el cual cuenta con un par de nuevos componente que se añaden a la arquitectura de *Storm*, estos componentes están diseñados para operar de forma distribuida geográficamente como también en distintos ambientes.

Los componentes se pueden separar en dos categorías, componentes de monitoreo y componentes de organización. Por parte de los monitores se cuenta con *QoSMonitor* que se localiza en cada supervisor de *Storm*, proveyendo *QoS awareness* al organizador que se localiza distribuido dentro de la arquitectura. Además, este se encarga de estimar la latencia de red con respecto a los otros nodos del sistema, por medio de la captación de información intra e inter-nodo, además de monitorear la disponibilidad del nodo y la utilización de sus recursos. El otro componente de monitoreo es *WorkerMonitor*, él es ejecutado en cada *worker*, capturando la velocidad de intercambio de los datos, ya sea saliente o entrante, de cada *executor* que se ejecuta en el *worker*, guardando esta información en una base de datos local que luego es usada por el organizador. La arquitectura de *Storm* junto con los componentes que se agregan se puede ver en la Figura 2.6.

Los componentes de organización son, *AdaptiveScheduler*, el cual se encuentra en cada supervisor de *Storm* y representa de forma distribuida al organizador, ejecutando las políticas de ubicación distribuida. El *AdaptiveScheduler* ejecuta un algoritmo de organización *QoS-aware* en cada *worker*, donde solo puede reasignar a los *executors* que ya están asignado a dicho *workers* para mejorar su desempeño. Este componente funciona con un solo *loop* de control de retroalimentación, el cual se ejecuta cada 30 segundos, siguiendo un ciclo MAPE, es decir, que posee cuatro etapas:

1. **Monitor:** el *AdaptiveScheduler* toma la información recolectada por *QoSMonitor* e identifica el conjunto de *executors* locales que pueden ser movidos, siendo un *executor* se considera movable si es que no está conectado con algún operador y puede ser instanciado por cualquier nodo de la red.

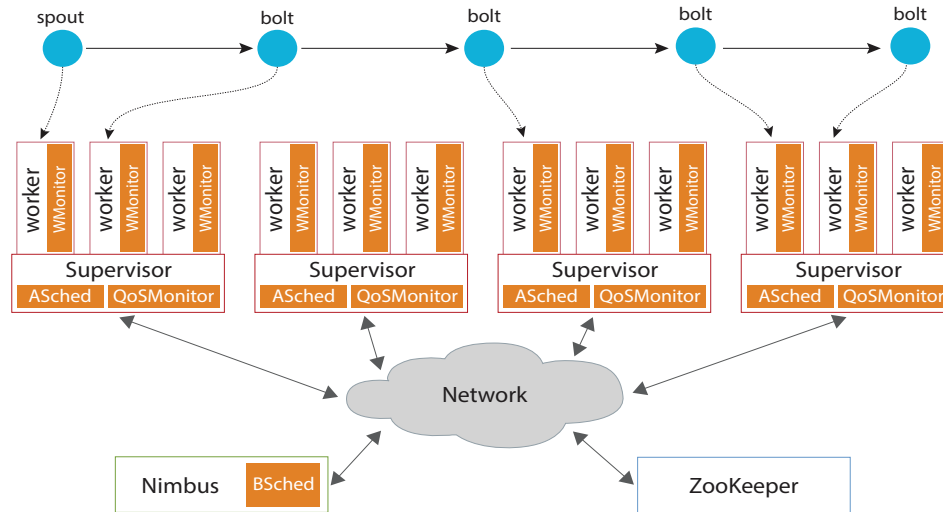


Figura 2.6: Arquitectura de *Storm* con nuevos componentes agregados
Fuente: [Cardellini et al., 2015].

Las siguientes fases se ejecutan secuencialmente por cada *executor*, debido a que de no ser de esta forma se da la posibilidad de que múltiples relocalizaciones podrían terminar siendo negativas para el desempeño del sistema. Para esto utiliza el acercamiento propuesto por Pietzuch [Pietzuch et al., 2006].

2. **Analizar:** determina si uno de los *executors* identificados como movibles se puede relocalizar de forma más eficiente. Esta etapa utiliza un algoritmo de costo-espacio, que calcula la distancia en un espacio multidimensional, tomando como las dos primeras dimensiones la latencia de la red y las otras dos como la disponibilidad y la utilización del nodo. Básicamente, busca en que posición puede ser reubicado de mejor forma el *executor* candidato.
3. **Plan:** Si es que se encuentra que un *executor* necesita ser reubicado, basándose en los resultados del algoritmo de costo-espacio, el algoritmo determina qué *worker* será el que va a ejecutar a dicho *executor*, y si es que no se encuentra uno, el ciclo MAPE termina. En caso de encontrarlo, se ejecutará un algoritmo de dos pasos, que consiste en dos pruebas, necesitando pasar ambas pruebas para poder ser reubicado. Primero, evalúa la “distancia relativa” para impedir el movimiento del *executor* si es que el cambio de posición no mejora el rendimiento del sistema. Si es que esta prueba es aprobada, se utiliza un algoritmo “look-ahead” para evitar que se creen asignaciones cíclicas, debido a que el algoritmo las puede producir. En el caso de que no se necesite reubicar al *executor* que sigue con el siguiente *executor* candidato.
4. **Ejecutar:** Se ejecuta el cambio y la decisión se comunica a través de *ZooKeeper*. El *executor* es detenido y la topología queda en un estado de *cooldown* por una cantidad determinada de ciclos del *AdaptiveScheduler*, impidiendo que cualquier *executor* pueda

ser reubicado, esto debido a que *Storm* no preserva los estados de los *executor* en la reasignación.

El último componente de monitoreo, *BootstrapScheduler*, es la versión centralizada de *QoS-Aware* que se ejecuta en *Nimbus*, encargada de asignar los *executors* si es que se pierde un *worker*.

2.1.3.2. *Spark*

Spark es un sistema de propósito general para computación en clúster, el cual está orientado a la velocidad y a proporcionar APIs de alto nivel en lenguajes como Java, *Scala*, *Python* y *R*, junto con un motor optimizado que soporta grafos de ejecución [Spark, 2016]. *Spark* utiliza un grafo acíclico dirigido (DAG) para representar las tareas que ejecuta en el clúster [MapR, 2015]. *Spark* soporta distintas herramientas incluyendo *Spark SQL* para el procesamiento de SQL y datos estructurados, *MLlib* para *machine learning*, *GraphX* para procesamiento de grafos, y *Spark Streaming* para procesamiento de *streams*. Este último se encarga de procesar información en tiempo real desde distintas fuentes. *Spark* utiliza un tipo de dato llamado RDD, el cual permite la segmentación del *stream* por intervalos de tiempo. Los RDD son enviados y procesados por el motor de *Spark* en lotes, lo cual lo diferencia de otros SPS, esto debido a que busca un mejor manejo de los posibles fallos.

2.1.3.3. *Flink*

Flink es un *framework* y un motor de procesamiento distribuido para computación con estados sobre *streams* de información, es decir, que el sistema tiene en cuenta múltiples eventos juntos para poder crear sus salidas [Friedman y Tzoumas, 2016]. Los *streams* pueden ser procesados como dos tipos de *streams*, *unbounded* o *bounded streams*, el primero es aquel al cual no se le define un fin, estos deben ser procesados de forma continua, y generalmente requiere que los eventos sean consumidos en un orden específico, como lo es el orden en que ocurren. Los *streams* de tipo *bounded* son aquellos que se les define un comienzo y un fin, estos pueden ser procesados luego de que toda la información sea consumida sin importar el orden, debido a que estas pueden ser ordenadas. Su procesamiento se conoce también como *batch processing*.

Apache Flink, permite procesar estos dos tipos de *streams*, debido a que cuenta con un control preciso del tiempo y los estados de los *unbounded streams*. Además, pueden procesar los *bounded streams* de forma interna a través de algoritmos y estructuras de datos que están diseñadas específicamente para conjuntos de tamaño fijo de datos.

La gran diferencia de *Flink* con otros SPS, es la utilización de ventanas (*windows*), las cuales le permite tener control sobre la cantidad de datos que está procesando, estas ventanas pueden ser ventanas de tiempo (por ejemplo, 30 segundos) o ventanas de datos (por ejemplo,

cada 100 eventos). Debido a esto, se debe tener un fuerte control respecto a los tiempos que influyen en el sistema, para esto hacer una separación en distintos tipos estos son:

- **Tiempo de Evento:** es el tiempo en que un evento es creado, usualmente identificado por una etiqueta de tiempo en cada evento.
- **Tiempo de consumo:** es el tiempo en que el evento llega a *Flink*.
- **Tiempo de procesamiento:** es el tiempo local de cada operador que realiza una operación basada en tiempo.

Gracias a esto, *Flink* puede procesar *streams* tanto como un SPS como *batch*, sin importar que el tiempo en que los eventos llegaron no sea concurrente respecto a el tiempo en que realmente fueron generados por sus fuentes, consiguiendo un resultado más preciso y además, le permite entregar resultados parciales.

2.1.3.4. *Simple Scalable Streaming System (S4)*

S4 es un SPS inspirado por el modelo de *MapReduce*, permitiendo procesar miles de consultas por segundos a una baja latencia [Neumeyer *et al.*, 2010b]. Su arquitectura esta basada en un modelo de actores [Agha, 1985], lo que le permite ser un sistema altamente concurrente mientras se mantiene una interfaz simple. Su diseño esta basado en el traspaso de mensajes, utilizando un grafo de procesamiento, donde los operadores son llamados elementos de procesamiento (PE). Además, S4 utiliza un modelo *push*, el cual consiste en que los eventos son enviados al siguiente PE lo más pronto posible y si en caso que el *buffers* de uno de los PE se fuera a llenar, los eventos enviados son omitidos por él, lo que lleva a pérdida de eventos.

Por el lado de la tolerancia a fallas, S4 provee recuperación de estados por medio de un *checkpointing* descoordinado, en donde un nuevo nodo toma el lugar del nodo que ha falla, volviendo a empezar desde el último punto de *checkpoint* del estado. Debido a esto los eventos recibidos entre el último *checkpointing* y la recuperación del nodo, se pierden. Actualmente S4 se encuentra retirado de la incubación Apache [Foundation, 2014].

2.2. **MOBILE COMPUTING y FOG COMPUTING**

Mobile computing es la capacidad que poseen los dispositivos móviles para realizar procesamiento, pero estos dispositivos, a diferencia de los computadores normales, poseen ciertos recursos limitados, como lo es la duración de la batería, el rango de señal, etc. Estas limitaciones sumadas a la heterogeneidad de los recursos computacionales que poseen los dispositivos móviles conducen al surgimiento de arquitecturas de procesamiento de información, estas son, *mobile grid computing* y *mobile cloud computing*, además de *fog computing*.

2.2.1. MOBILE GRID COMPUTING

La arquitectura de *mobile computing* busca aplicar el modelo tradicional de *grid computing* [Foster *et al.*, 2008] para dispositivos móviles, de forma de crear clúster que utilicen dispositivos móviles para realizar alto procesamiento de información. Un prototipo de esta arquitectura es DroidCluster [Busching *et al.*, 2012], donde se muestra la factibilidad de este tipo de arquitectura utilizando seis dispositivos Android conectados a un punto de acceso.

Una implementación real de *mobile grid computing* es NativeBOINC [Szpakowski, 2012], que es un cliente que utiliza también Android, el cual permite a los usuarios suscribirse a distintos proyectos de investigación para así compartir los recursos de procesamiento de sus dispositivos móviles, ya sea tanto CPU como GPU. Un proyecto similar es HTC *Power to Give* ⁴, cuyo objetivo es contribuir con poder de procesamiento, generado por dispositivos móviles, a proyectos de investigación en diversas área, como medicina, ciencias o causas medioambientales.

Una de las preocupaciones principales en cuanto se busca utilizar dispositivos móviles en forma de clúster es la limitada disponibilidad de energía con la que cuentan, este costo energético es estudiado en [Comito *et al.*, 2017] cuyo enfoque se basa en el balance de la carga en la asignación de tareas dentro de una red inalámbrica de dispositivos móviles, la cual se realiza de forma eficiente para buscar maximizar el número de dispositivos conectados a dicha red.

2.2.2. MOBILE CLOUD COMPUTING

Al contrario de *mobile grid computing*, *mobile cloud computing* busca tomar ventaja de los recursos ilimitados que proporciona la nube para así suplementar la falta de recursos de los dispositivos móviles, ya sea, la capacidad de procesamiento, de almacenamiento o de batería. Busca realizar el procesamiento y almacenamiento de la información fuera de los dispositivos móviles [Dinh *et al.*, 2013], disminuyendo así la cantidad de energía utilizada por el dispositivo, como también permitir una mayor capacidad de procesamiento a dispositivos que no cuentan con ella.

Las ventajas de *mobile cloud computing*, son el mejoramiento de la utilización de la batería, incremento de la capacidad de almacenamiento individual de los dispositivos y las capacidades de procesamiento, además, de la sincronización de aplicaciones en múltiples dispositivos, entre otras. Esto lleva un costo asociado a la necesidad de estar siempre conectado a Internet, incrementando el uso de energía por ese lado. En relación a esto se ha propuesto estrategia para poder transmitir datos de manera inteligente hacia la nube, teniendo en cuenta el estado de red [Shu *et al.*, 2013].

⁴<http://www.htc.com/es/go/power-to-give/>

2.2.3. FOG COMPUTING

Otro paradigma que surge es *Fog Computing* [Bonomi *et al.*, 2014] o computación en la niebla, este surge debido a la observación de que los dispositivos que se encuentra en el borde de la red son los mismo que generan la información procesada, hablese de *smartphones*, *tablets*, sensores de distinto ámbito, componentes de IoT o domótica. Estos dispositivos poseen una capacidad de procesamiento, algunos hasta igualan a la capacidad de un servidor pequeño que se encuentra en la nube. Por tanto, se busca utilizar estos recursos, ya sean de procesamiento o almacenamiento, etc. De por sí, este paradigma es más general, y esta pensado principalmente en la distribución geográfica de los recursos de computación. De esta última idea, es de donde aparece el argumento para el desarrollo del modelo con que se trabajará en este documento.

2.2.4. PROCESAMIENTO DE STREAM SOBRE DISPOSITIVOS MÓVILES

Existen varios acercamientos a utilizar sistemas de procesamiento de *stream* sobre arquitecturas compuesta por clúster de dispositivos móviles, estas propuestas presentadas aquí. De aquí sale conceptos que luego se verán utilizados en la presentación del actual modelo de procesamiento de *stream* asistido por dispositivos móviles, como también sirven de base en los cambios que se presentan como objetivo de este documento.

2.2.4.1. MobileStorm

MobileStorm[Ning *et al.*, 2015] es un SPS que opera sobre dispositivos móviles. Está basado en el modelo de procesamiento de *Storm*, es decir que realiza su trabajo de manera concurrente y de forma distribuida a través de múltiples dispositivos. Solo implementa dos tipos de políticas de planificación de tuplas, *shuffle grouping* o local, la ventaja de la última es que permite reducir el consumo de batería evitando usos innecesario de la red. Junto con esto, *MobileStorm* se aleja de la representación lógica de un evento en *Storm*, dejando de ser un objeto Java serializado para pasar a ser uno de formato libre, debido a que los dispositivos móviles no están optimizados para poder serializar y deserializar ese tipo de objetos.

Los componentes *Nimbus* y *Zookeeper* son desplegados en la nube debido a su alta utilización de recursos. Cada nodo dentro del clúster de dispositivos móviles ejecuta un *Worker Node*, a su vez cada uno de estos ejecuta los *executors* que se encargarán de realizar las tareas asignadas por *Nimbus*. Más que generar un apoyo a la arquitectura que posee *Storm*, *MobileStorm* pretende ser una alternativa a este, a diferencia del modelo que se presenta en este documento.

2.2.4.2. Symbiosis

Symbiosis[Morales *et al.*, 2014] es una arquitectura distribuida orientada al procesamiento en tiempo real de la información, través de *smartphones* o *tablets*. Esta arquitectura genera pequeños clústers en donde los nodos son los dispositivos móviles, los que pueden tanto generar información, como procesarla. Para esto, cuenta con un *Processing Access Point* (PAP), el cual se encarga de distribuir los *streams* entre los nodos. La información procesada localmente se envía a un servidor global que se encarga de organizarla. Por otro lado, el sistema se hace cargo de problemas como pérdida de señal, salidas abruptas del sistema por parte de los dispositivos u otro suceso que pueda provocar pérdida de información para el sistema. Para solucionar esto, *Symbiosis* utiliza un modelo de *checkpointing* que se gatilla en función del estado de la conexión y el estado de la energía disponible que tiene el dispositivo, permitiendo almacenar el estado de los operadores antes que la información se pierda, pudiendo ser recuperada fácilmente.

2.3. ACTUAL MODELO DE PROCESAMIENTO DE STREAM ASISTIDO POR DISPOSITIVOS MÓVILES

El modelo presentado en [Morales, 2016] es un modelo que administra los recursos de los dispositivos móviles para asistir a la arquitectura de un SPS, para esto se presenta una arquitectura con distintos componentes que se asocian a la arquitectura a nivel físico del SPS como también de los dispositivos móviles, y una parte lógica que se encarga de administrar dichos componentes para incrementar el poder de computo del SPS que se está asistiendo.

2.3.1. ARQUITECTURA

La arquitectura del modelo se basa en capturar los recursos de dispositivos móviles que se encuentren semi-estáticos, es decir, dispositivos que se encuentren en lugares como estadios, aeropuertos o el transporte público, donde la movilidad del dispositivo sea relativamente baja. En la Figura 2.7 se puede observar un ejemplo de algunas de las áreas en donde se pueden encontrar dispositivos que pueden ayudar con su poder de procesamiento a un SPS, a estas áreas se les denomina regiones de procesamiento. Cada región es construida con los dispositivos que están conectados a un punto de acceso en común, donde el clúster principal de un SPS puede recaudar el poder de más de una de estas regiones, interactuando con ellas de forma independiente.

Para definir las topologías de los grafos de procesamiento el modelo se basa en *Storm*, el cual llama *spouts* a los operadores fuente y *bolts* a los operadores de procesamiento y a los operadores terminales. Al igual que en *Storm*, los operadores reciben eventos desde el *upstream* para luego enviarlos hacia el *downstream* del grafo de procesamiento.

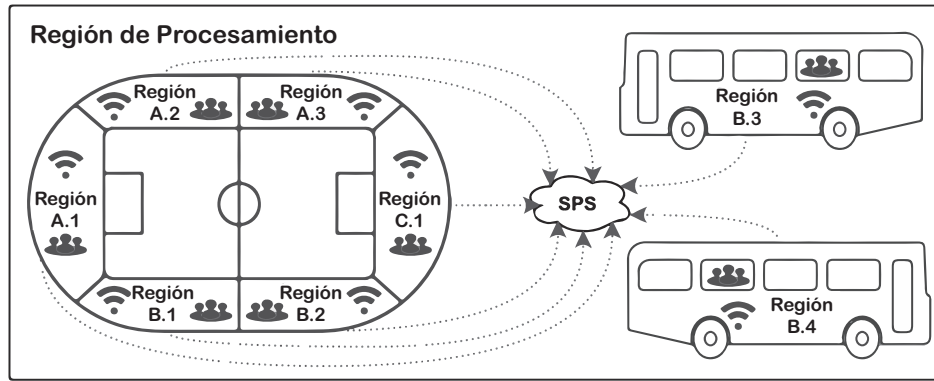


Figura 2.7: Ejemplo del concepto de las distintas regiones de procesamiento
Fuente: [Morales, 2016].

Los nodos físicos actúan como contenedores de los operadores, replicando su funcionamiento dentro de la topología. Estos nodos físicos se diferencian entre nodo móvil y nodo estático, donde un nodo estático es aquel nodo normal de un SPS que está desplegado en un clúster y cuenta con una conexión permanente a la electricidad y al Internet, por lo que su persistencia está asegurada. En cambio, un nodo móvil es aquel que está asociado a un dispositivo móvil y se encuentra entregándole poder de procesamiento al SPS.

Para mantener persistencia dentro del modelo, los nodos móviles pueden alojar solo a los operadores de procesamiento. Además, cabe destacar que un nodo móvil siempre debe tener un nodo estático como réplica de respaldo debido a la necesidad de tolerancia a fallas.

Finalmente, dentro de cada región se definen los siguientes componentes:

- **Region Manager (RM):** este componente se encuentra alojado junto con *Nimbus* en un nodo estático, el cual coordina y monitorea a los operadores dentro de la región.
- **Operator Manager (OM):** es el encargado de la comunicación entre el operador y el RM, encontrándose presente en los nodos físicos, principalmente en los nodos móviles, que alojan a los operadores de procesamiento, permitiéndole tanto administrar el balance de carga dentro de este operador como capturar datos sobre los recursos de este nodo, para administrar sus recursos.

En la Figura 2.8 se puede ver como estos componentes se integran a la arquitectura que posee *Storm*, donde se le agrega un alias a los *workers* dependiendo si están siendo ejecutados dentro de un nodo estático o un nodo móvil.

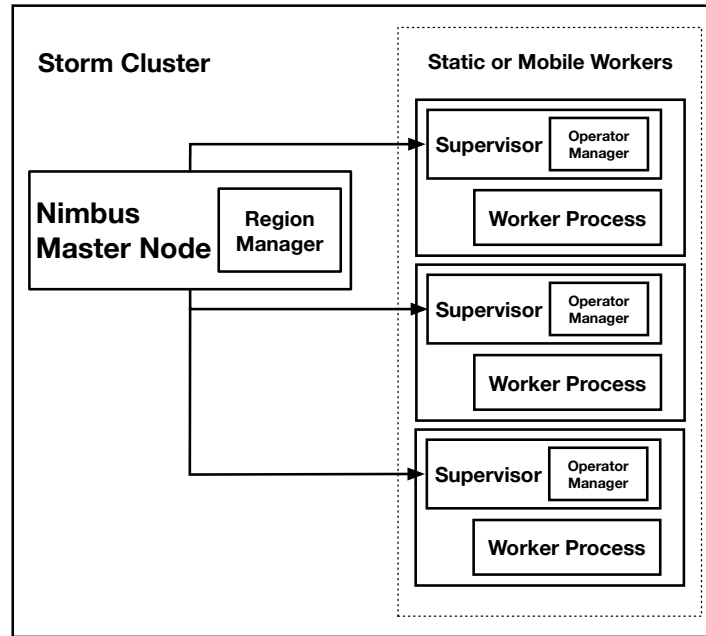


Figura 2.8: Arquitectura de *Storm* con la integración de los componentes del modelo
Fuente: [Morales, 2016].

2.3.2. FORMACIÓN DEL CLÚSTER Y MANTENIMIENTO DE LA REGIÓN

El modelo de integración se basa en un modelo para sistemas autoadaptable, llamado MAPE, el cual está comprendido por cuatro fases: Monitorear, Analizar, Planear y Ejecutar. Este modelo de integración utiliza este ciclo en dos niveles como se presenta en la Figura 2.9, donde el nivel más alto es ejecutado por el RM para formar y mantener el clúster con los nodos disponibles. Mientras que, en el nivel más bajo, ejecutado por cada OM, se encarga de monitorear al nodo de forma local.

El RM ejecuta un proceso durante la ejecución del programa el cual recibe solicitudes de admisión por parte de los dispositivos móviles, a lo que el RM crea un nodo móvil con su correspondiente OM en dicho dispositivo. El RM solicita el estado del dispositivo móvil, es decir, la intensidad de señal y su batería restante. El modelo configura y distribuye los nodos móviles de manera automática, el RM ejecuta este proceso mediante el ciclo de control MAPE durante un tiempo ΔT , donde cada uno de los pasos son:

- **Monitorear:** El RM recolecta los estados de los recursos de cada nodo móvil, donde estos nodos se comunican con el RM de forma asincrónica y en cada ciclo la fase de monitoreo solo considera a aquellos nodos que actualizaron su información.
- **Analizar:** El RM procesa la información recolectada en la fase anterior a través del Algoritmo 2.1, donde se determina el estado de los recursos críticos de cada nodo móvil obteniendo una lista de los nodos móviles categorizados por sus recursos entre

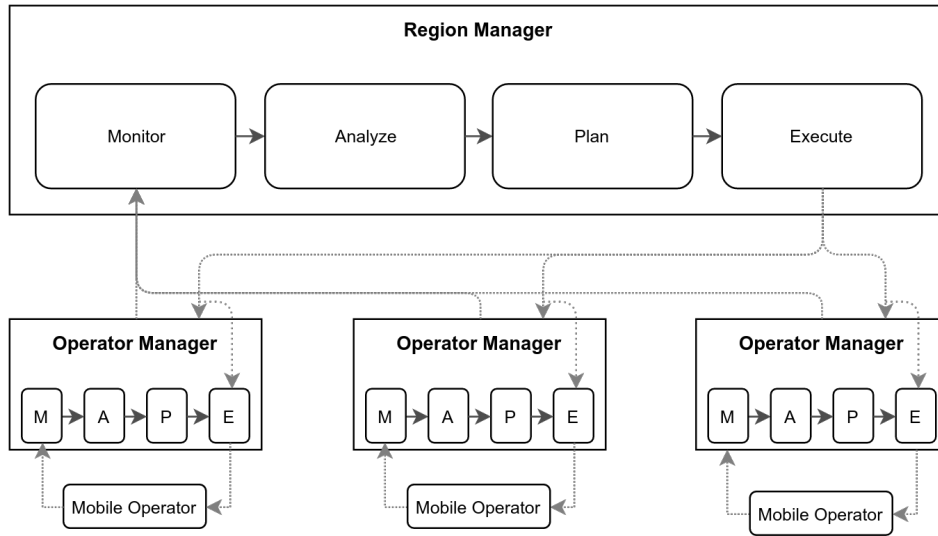


Figura 2.9: Modelo MAPE de dos niveles
Fuente: [Morales, 2016].

“normal” o “baja” comparado contra un umbral predefinido.

Algoritmo 2.1 Determinación de estado de recursos críticos.

Entrada: *lista_mop* ← Lista de nodos móviles

Salida: Lista de nodos móviles con información procesada

```

1: for each mn in lista_mn do
2:   if mn.energía > umbral_energía then
3:     mn.energía = “normal”
4:   else
5:     mn.energía = “baja”
6:   end if
7:   if mn.señal > umbral_señal then
8:     mn.señal = “normal”
9:   else
10:    mn.señal = “baja”
11:  end if
12: end for
13: return lista_mn

```

- **Planificar:** El RM agrega a los nodos móviles a dos listas dependiendo de la categorización que obtuvo en el paso anterior. Agregando a los nodos que poseen sus recursos con un estado “normal” a la lista de activación y agrega a aquellos nodos que no cumplen esta condición a la lista de limpieza, debido a que el uso de estos dispositivos con niveles riesgosos de intensidad de señal o batería pueden llevar a pérdida de eventos.

En caso contrario, se les permite a los nodos en la lista de activación replicar operadores de procesamiento, indicándoles a cual de estos operadores de procesamiento deberá replicar basado en un algoritmo de *round-robin*. Para formar esta lista se genera un mapa virtual de los nodos de móviles presentes que replican a los operadores de procesamiento, consiguiendo una vista de todos los recursos móviles disponibles. Por último, cabe destacar que el mapa virtual no considera a los operadores estáticos, puesto estos son administrados de forma transparente por el SPS.

- **Ejecutar:** Usando la lista de limpieza y la lista de activación, el RM envía a los OM las órdenes de, ya sea, remover el operador o crear uno nuevo. Finalmente, el RM envía la información global del sistema a todos los OM para que estos estén informados de la localización de los operadores dentro de la topología.

2.3.3. FLUJO DE EVENTOS Y BALANCE DE CARGA

El modelo incluye un modelo *pull* para el envío de eventos hacia los nodos móviles, esta decisión se basa en las limitaciones de estos operadores, dejando sin interferir el envío desde y hacia los nodos estáticos, lo cual queda bajo las políticas de envío propias del SPS. Este modelo *pull* funciona a través de una cola de eventos, la cual se encontrará en el nodo estático, de donde los nodos móviles sacarán los eventos para ser procesados. Esta cola es de tamaño fijo y posee un límite de tiempo para desencolar los eventos, cuyo tiempo es pequeño para no perder las características de tiempo real del SPS. Cumplido este tiempo si el evento no es procesado será enviado a hacia los nodos estáticos que replican al mismo operador que el nodo móvil. Cabe notar que este modelo permite a los nodos móviles puedan procesar los eventos a la medida de sus capacidades, evitando sobrecarga en estos nodos móviles. Finalmente, cada nodo móvil solo puede interactuar con un nodo *upstream*, debido a que un excesivo número de mensajes entre los OM congestionaría la red y repercutiría en el consumo energético de los dispositivos móviles.

Para controlar el consumo excesivo de los recursos del dispositivo móvil, considerando que estos deben seguir siendo utilizados mientras prestan su poder de procesamiento al SPS, el modelo incorpora una política de balance de carga reactiva que se implementa en el segundo nivel del ciclo MAPE, como se muestra en la Figura 2.9. Al igual que el nivel anterior, este ciclo tiene un periodo de $\Delta T'$ y es ejecutado en el OM de cada nodo móvil. Las etapas del ciclo son:

- **Monitorear:** al cumplir el ciclo $\Delta T'$ el OM consulta la utilización del CPU del nodo móvil y la envía a la etapa de análisis.
- **Analizar:** con la información obtenida en la fase anterior, el OM ejecuta el Algoritmo 2.2 para saber si es que el nodo móvil se encuentra en estado saludable o sobrecargado, el cual se determina respecto a un umbral definido.

Algoritmo 2.2 Análisis de carga de nodos móviles.**Entrada:** $utilización_cpu \leftarrow$ Utilización de CPU del nodo móvil**Salida:** Estado del nodo móvil

```

1: if  $utilización\_cpu > umbral\_utilización\_cpu$  then
2:    $estado = "sobrecargado"$ 
3: else
4:    $estado = "saludable"$ 
5: end if
6: return  $estado$ 

```

- **Planificar:** si es que el OM determina que el nodo se encuentra sobrecargado se comunica con el OM del nodo *upstream* para que este aplique un algoritmo de planificación *fast-adaptation-weighted-round-robin* entre el nodo sobrecargado y un nodo estático como se observa en la Figura 2.10, donde se ve que ambos nodos comienzan con pesos iguales W_x en un ciclo $i + 1$. Si en el siguiente ciclo el nodo móvil sigue sobrecargado, el OM se comunica nuevamente para que el OM del nodo *upstream* aumente el peso del nodo estático a W_y , donde este proceso se repetirá hasta que el nodo móvil vuelva a un estado saludable.

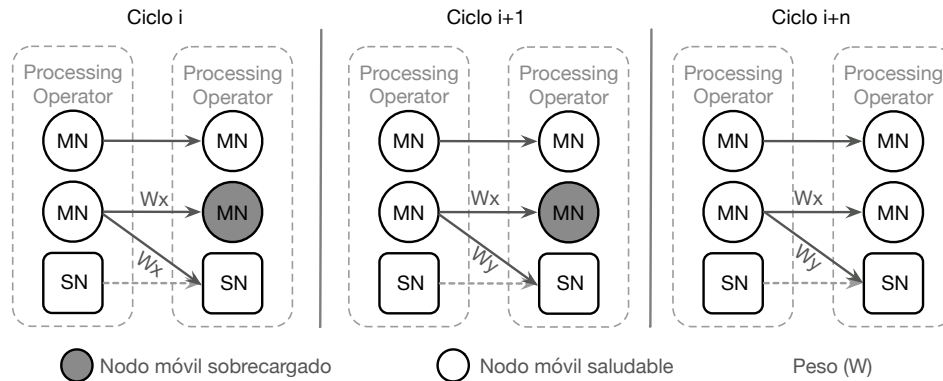


Figura 2.10: Proceso de balance de carga entre nodos móvil y un nodo estático

Fuente: [Morales, 2016].

- **Ejecutar:** se envía el mensaje al OM del operador *upstream* para que inicie la regulación de los pesos según sea el caso.

El diseño del modelo es flexible permitiendo utilizar eventos de manera individual o en forma de lotes, empaquetándolos para enviarlos, y luego, desempaquetándolos para que puedan ser procesados, considerando cada lote como un evento individual. La decisión tras esto es debido a que ayuda a bajar el consumo energético de los dispositivos móviles cuando se envían muchos eventos de tamaño pequeño, disminuyendo las conexiones necesarias para enviar y recibir estos eventos.

2.3.4. TOLERANCIA A FALLAS

El modelo detecta y repara el SPS si es que el RM no recibe el reporte de cada OM, asumiendo que el nodo no está disponible. En el caso de los OM, si es que detecta que uno de los nodos *downstream* a fallado cambia el proceso de organización para el envío de eventos, para que estos sean mandados al nodo estático que replica al mismo operador que el nodo fallido. Como se muestra en la Figura 2.11, se puede ver que al detectar una falla el OM envía los eventos hacia un nodo estático *downstream* y otro nodo móvil *downstream* empieza a recibir los eventos desde dicho nodo estático.

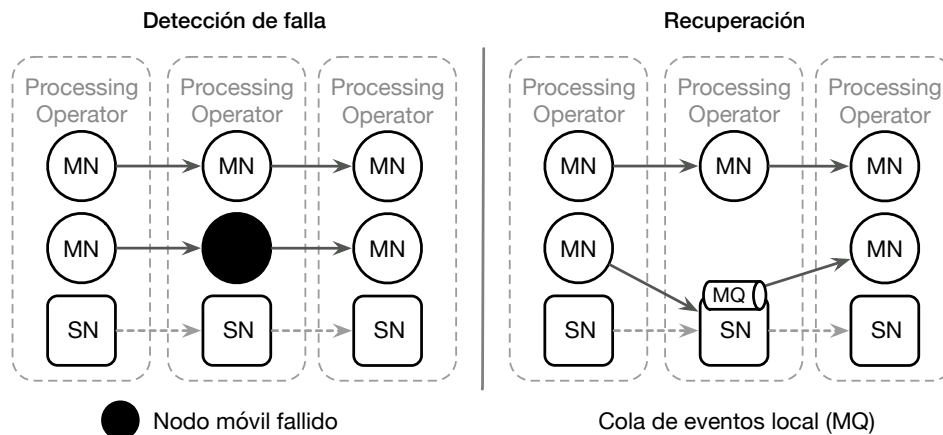


Figura 2.11: Ejemplo del comportamiento del modelo cuando un nodo móvil falla
Fuente: [Morales, 2016].

Finalmente, se utiliza un modelo de *checkpointing* basado en [Morales et al., 2014] para disminuir la pérdida de eventos debido a la pérdida de señal o batería, realizando el *checkpoint* cuando el estado de los recursos es crítico, el cual es enviado a la réplica del operador desplegada en un nodo estático.

2.4. SIMULACIÓN DISCRETA POR EVENTOS Y NS-3

2.4.1. SIMULACIONES

Las simulaciones son imitaciones del mundo real, ya sea de procesos o sistemas, a través del tiempo. Una simulación es generar una historia artificial de un sistema, permitiendo observar esta historia para poder inferir sobre las características operativas que tendría el sistema real que se está representando. Las simulaciones son una metodología indispensable para resolver problemas de la vida real, siendo usadas para describir y analizar el comportamiento de los sistemas, preguntando “¿Y sí?” sobre sistemas reales y ayudando al mejoramiento del diseño de estos, dando la posibilidad de modelar sistemas tanto reales como conceptuales [Banks, 1998].

Un modelo es la representación de un sistema, este debe ser lo suficientemente complejo para permitir estudiar el problema que se busca resolver, pero no lo tanto como para escapar de lo que se busca responder, por lo que es necesario definir límites y fronteras de lo que el modelo debe representar del sistema.

Existe distintos tipos de simulaciones, una de estas son las simulaciones estadísticas, como lo es el método de Montecarlo. Otras son las discretas y las continuas, estas últimas se caracterizan por funcionar a través de una función continua de números reales representando un cambio continuo en el estado del sistema. En cambio, las simulaciones discretas se caracterizan por basarse en fenómenos contables, siendo las simulaciones discretas por eventos las que se basan en un número finito de eventos que ocurren de forma dinámica, es decir, que el paso del tiempo juega un aspecto crucial en el sistema.

2.4.2. SIMULACIÓN POR EVENTOS DISCRETOS

Las pruebas realizadas para este documento fueron realizadas dentro de ambientes simulados debido a que permite realizar estas de forma más fácil, además controlar de esta misma forma las variables que afectarán a estas. Se destaca principalmente a ns-3 debido a que es la herramienta que se eligió para realizar dichas pruebas debido a si simulación de eventos discretos que permite simular la transmisión y recibimiento de tuplas del sistema de procesamiento de *streams*.

2.4.2.1. Principios de simulación

Existen varios conceptos subyacentes a una simulación, conceptos como modelo, eventos, entidades y atributos, variable de estado del sistema, actividades y retrasos, entre otros.

Los **eventos** son sucesos que ocurren en un sistema, el cual cambia el estado de dicho sistema. Existe dos tipos de eventos, externos e internos, en donde los eventos externos son las ocurrencias que suceden desde fuera de la simulación y los eventos internos son los que ocurren dentro de la simulación [Houlding, 2012]. Los eventos son almacenados en lista llamada *future event list* (FEL) como *event notices*, donde cuentan con una estructura de (*tiempo*, *tipo*). En la Figura 2.12 se ve la evolución de los eventos a medida que pasa el tiempo. Otros conceptos importantes para una simulación son:

- **Variables de estado del sistema:** estas variables son las que poseen la información necesaria para saber el estado actual del sistema, permitiendo saber qué pasó en un sistema en un punto dado del tiempo. Estas variables se determinan teniendo en cuenta qué es lo que se va a estudiar del sistema, por lo que una colección de variables para distintos estudios puede ser distinta para el mismo sistema real, lo que hace que determinar que variables son necesarias para cada experimentación se considere tanto un arte como una ciencia, lo que conlleva a la creación de heurísticas u otros métodos.

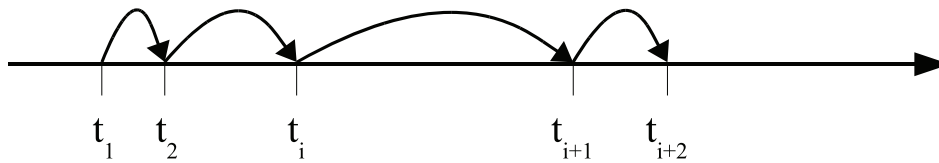


Figura 2.12: Durante la simulación el sistema solo cambia de estado en puntos discretos de tiempo t_i .

Fuente: [Wehrle *et al.*, 2010].

- **Entidades y Atributos:** las entidades son la representación de objetos que necesitan una definición explícita, las cuales pueden ser dinámicas, es decir, que se mueve a través del sistema, o pueden ser estáticas que solo sirven a otras entidades. Cada entidad posee atributos que solo le pertenecen a ella, lo que hace que se consideren como valores locales, y al igual que las variables de estado puede que los atributos que sean importantes en un sistema no sean necesarios en otro.
- **Recursos:** un recurso es un ente que sirve a las entidades dinámicas, estos puede servir a una o más entidades al mismo tiempo, sirviendo uno o más unidades del recurso. Puede estar en muchos estados posibles, pero generalmente como mínimo sus posibles estados contienen, “ocupado” y “libre”. Si es un recurso rechaza la petición de una entidad debido a que esta ocupado esta puede unirse a una cola de espera o tomar otra acción.
- **Lista de procesamiento:** las entidades son administradas a través de ser asignadas a los recursos, lo cuales las colocan en colas de forma ordenada, generalmente siendo servidos de forma FIFO o LIFO, entre muchas otras.
- **Actividades y retrasos:** una actividad es una duración de tiempo, la cual se sabe de antemano cuanto es el tiempo específico que debería durar, por lo que permite registrar cuando terminará al momento de ser iniciada. Esta duración puede ser constante, generada aleatoriamente por una distribución estadística, o puede ser calculada en base al estado del sistema. Un retraso es una duración de tiempo indefinido causada por una combinación de condiciones del sistema. Cuando una entidad se agrega a una cola, el tiempo que estará en la cola no se sabe debido a que ese tiempo depende de otros eventos en el sistema.

2.4.2.2. Definición de simulación discreta por eventos

Las simulaciones discretas por eventos son aquellas que son representadas por una finita secuencia de eventos, en donde el estado del sistema solo se cambia en los instantes de tiempo en que un evento ocurre, describiendo cada evento de forma discreta, los cuales ocurren como consecuencia de una actividad o un retraso.

Las entidades compiten por recursos del sistema, con la posibilidad de unirse a colas mientras esperan por la liberación de recursos, además, las entidades pueden ser retenidas por

actividades o retrasos por una cierta cantidad de tiempo. El sistema se va moviendo a través del tiempo por medio de algún mecanismo que simula el avance de este, actualizando así el estado del sistema por cada evento que ocurre en ese tiempo junto con liberación y captación de recursos que desencadene cada evento.

2.4.2.3. Algoritmo para el avance de tiempo de la programación de eventos

Este algoritmo es el principal para una simulación de eventos. Durante una simulación los estados van cambiando a medida del tiempo, por lo que es necesario que exista un reloj que se encargue de llevar el tiempo de simulación. Por otro lado, la lista de eventos futuros (FEL) contiene todos los eventos de forma ordenada respecto al tiempo de ocurrencia de ellos. En la Figura 2.13 se observa el diagrama de flujo de este algoritmo.

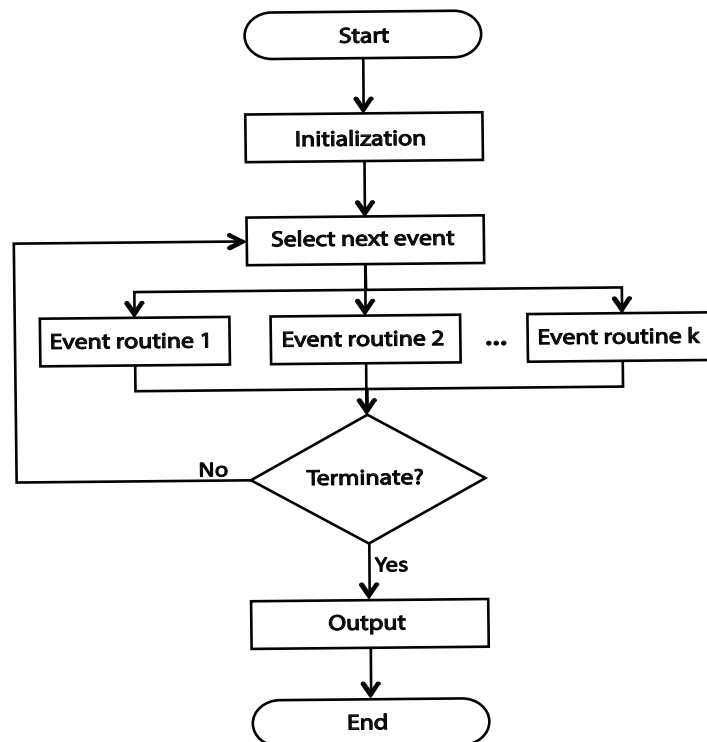


Figura 2.13: Diagrama de flujo del algoritmo para el paso de tiempo de una simulación de eventos discretos.

Fuente: [Wehrle *et al.*, 2010].

2.4.3. ns-3 (Network Simulator version 3)⁵

Debido a que las redes de computadores crecen cada vez más, tanto en complejidad como en tamaño, la necesidad poder simular estas redes de forma precisa y escalable se hace cada

⁵<https://www.nsnam.org/>

vez más crítica. Con estudios basados en simulaciones estas redes pueden ser estudiadas en detalles teniendo en cuenta distintas escalas, con distintos datos, y distintas condiciones, produciendo resultados reproducibles y analizables.

Para esto, se creó la serie de herramientas llamadas *ns*, las cuales son simuladores de eventos discretos que permiten simular redes de computadores, es principalmente usado para educación e investigación. Actualmente, la versión que es mantenida y sigue en desarrollo es *ns-3*, desarrollo liderado por Tom Henderson, George Riley, Sally Floyd y Sumit Roy en el año 2006 [Henderson, 2006], el cual fue lanzado en 2008 [Henderson et al., 2008].

ns-2, el predecesor de *ns-3*, fue el estándar de las investigaciones académicas en protocolos de redes y métodos de comunicación, pero a pesar del gran número de publicaciones que utilizaron *ns-2*, fue necesario un nuevo proyecto, debido a que este tomaba en cuenta la visión de cómo modelar redes de forma que se acomode de mejor manera a la investigación de redes [Riley y Henderson, 2010]. Bajo esta motivación es que surge *ns-3*, que además cuenta con otros cambios:

- **Realismo:** El mayor enfoque del cambio, lo que permite mayor reutilización y ayuda a disminuir la divergencia de los resultados experimentales. Para esto se modificaron los nodos de Internet de forma que represente de mejor manera a los reales, incluyendo soporte para *sockets* o dispositivos de red, además, del uso de protocolos IP, entre otros.
- **Nuevo núcleo de software:** fue diseñado para mejorar la escalabilidad, modularidad, entre otros. Este núcleo está escrito en C++ con una interfaz opcional en *Python*, además de agregar distintos patrones de diseños.
- **Arquitectura de *tracing*:** utilizando un diseño basado en *callbacks*, *ns-3* posee un *framework* que permite seguimiento y recopilación de estadísticas, permitiendo personalizar los datos de salidas sin necesidad de modificar el núcleo de la simulación.

2.4.3.1. Flujo de trabajo

El proceso general para crear una simulación en *ns-3* es:

1. **Definición de la topología:** instanciar los objetos que se necesitarán y las relaciones que tendrán entre ellos.
2. **Desarrollo del modelo:** modelos son añadidos a la simulación. Por ejemplo, protocolos UDP, IPv4, etc.
3. **Configuración de enlaces y nodos:** definir los valores por defecto que tendrán los modelos.

4. **Ejecución:** la simulación genera los eventos y los datos requeridos por el usuario son guardados.
5. **Análisis:** después que la simulación terminó, los datos estarán disponibles con estampillas de tiempos, permitiendo que estos datos puedan ser procesados y analizados de forma estadística.
6. **Visualización gráfica:** data pura generada por ns-3 o datos procesados pueden ser graficados para el mejor entendimientos y presentación de estos.

2.4.4. OTROS SIMULADORES

Junto con ns-3 existen una gran cantidad de simuladores de redes, cada uno con distintas ventajas y desventajas, por lo que la decisión de elegir uno por sobre otro se basa en las características de lo que se va a simular y los resultados que se desean obtener [Siraj *et al.*, 2012]. Entre estos simuladores destacan OMNet++ y Opnet Modeler.

OMNeT++⁶ es una extensiva, modular librería basada en componentes de C++, que también posee un ambiente gráfico y de desarrollo. Fue creada con el fin de crear simulaciones de redes de comunicación y sistemas distribuidos, pero en vez de ser un simulador especializado, fue diseñada de la forma más general posible. De forma que solo incluye herramientas básicas de simulación y son los módulos los que permiten agregar herramientas más específicas al campo de simulación de redes.

Opnet Modeler (OPTimized Network Engineering Tool)⁷ es una herramienta de modelado y simulación de redes de comunicación ampliamente utilizado. Este permite estudiar y diseñar redes, dispositivos, protocolos y aplicaciones, brindándoles escalabilidad y flexibilidad.

⁶<https://omnetpp.org/>

⁷<http://opnetprojects.com/>

CAPÍTULO 3

REDISEÑO DEL MODELO DE PROCESAMIENTO DE STREAM ASISTIDO POR DISPOSITIVOS MÓVILES

En esta sección se expondrán los cambios realizados al modelo de procesamiento de *stream* asistido por dispositivos móviles y los argumentos detrás de estos cambios. Estos, ya sea grandes o pequeños, se hicieron debido a aspectos que no se habían tomado en cuenta o falencias encontradas en los resultados presentados en [Morales, 2016].

3.1. MODELO DE TRANSMISIÓN DE EVENTOS

El actual modelo utiliza un modelo *pull* para realizar la transmisión de eventos hacia nodos móviles, ya sea desde un estático u otro nodo móvil. Un nodo *downstream* debe realizar una solicitud hacia el nodo *upstream* para que este responda enviando el evento, como muestra la Figura 3.14. En el caso de que el evento venga desde un nodo estático, esta solicitud se hace hacia una cola que contiene a los eventos por una cantidad de tiempo definida. Cumplido este tiempo el evento es retransmitido hacia un nodo estático, intentando de evitar sobrecargas a los nodos móviles y que estos puedan procesar eventos a medida de lo posible.

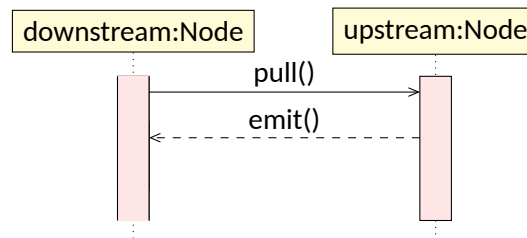


Figura 3.14: Ejemplo de la transmisión de un evento en el modelo actual

Fuente: Elaboración Propia.

Existen dos problemas con la utilización de este modelo. Primero, debido a que cada vez que un nodo *downstream* quiere procesar un evento debe realizar una solicitud y el nodo *upstream* responderla, aumentando al doble la cantidad de mensajes enviados para poder procesar los eventos, es decir, que por cada evento que un nodo necesite procesar debe enviar dos mensajes, lo que finalmente repercute tanto en la congestión del tráfico de mensajes dentro de la red del sistema como en la utilización de recursos de los nodos móviles, principalmente del uso de batería. Segundo, al tener esta cola de envío de mensajes va a existir una discrepancia entre los eventos que están en ella contra la capacidad de procesamiento del nodo *downstream* en un determinado momento, debido que va a existir un desfase entre estas dos dimensiones, lo que puede llevar a una saturación de estas colas y así aumentando la latencia de los eventos debido a que tienen que ser retransmitidos.

El cambio propuesto para el modelo de transmisión de eventos es utilizar un modelo *push*, el cual determina que cuando un evento es procesado por un nodo *upstream* este sea emitido inmediatamente, dependiente al algoritmo *fast-adaptation-weighted-round-robin*, hacia el nodo *downstream* correspondiente sin que este necesite solicitarlo, como lo muestra la Figura 3.15. Este evento llega a una cola en el nodo *downstream* donde espera ser procesado por el *worker* del nodo.

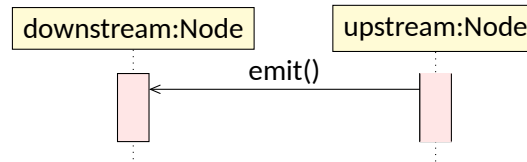


Figura 3.15: Ejemplo de la transmisión de un evento en el modelo push
Fuente: Elaboración Propia.

Este tipo de modelo ayuda a reducir la cantidad de mensajes enviados por cada evento procesado a la mitad, así reduciendo la utilización general de batería dentro del sistema. Por otro lado, debido a que los nodos *downstream* ahora reciben eventos sin ser solicitados podría levantar inquietud respecto a la sobrecarga de estos, pero se ve que el algoritmo de *fast-adaptation-weighted-round-robin* maneja de forma rápida y eficiente el flujo de eventos que se dirige hacia los nodos móviles para poder bajar la carga que es ejercida sobre un nodo que se pudiera encontrar sobrecargado.

3.2. TOPOLOGÍA DE LOS OPERADORES

Actualmente, el modelo plantea una restricción respecto a la cantidad de nodos *downstream* que están conectados a un nodo *upstream* móvil, ya que se intenta disminuir la cantidad de mensajes enviados por los nodos móviles, es decir, maximizar lo más posible la batería de un dispositivo respecto a la cantidad de eventos procesados. Debido a esta restricción los nodos móviles solo se pueden enviar eventos hacia un solo nodo a la vez.

El problema con esta restricción es que no considera que un nodo móvil puede replicar a un operador de procesamiento el cual se conecte a más de un operador dentro de la topología de procesamiento, es decir, que genere una bifurcación dentro del flujo de los eventos, como se muestra en la Figura 3.16. Esto lleva a que el nodo móvil que replica a dicho operador deba enviar un evento a por lo menos un nodo por cada operador *downstream* conectado, ósea que debe enviar como mínimo dos eventos por cada evento que procesa. Cabe diferenciar esto respecto al caso de que un nodo *downstream* se sobrecargue y se deba conectar el nodo *upstream* a un nodo estático. En este caso, el nodo *upstream* alterna el envío de eventos entre el nodo móvil y el nodo estático, por lo que siempre mandara un evento a la vez y no dos como es en el caso de la bifurcación.

Para solucionar este problema, se agrega otra restricción adicional al modelo, la cual se hace

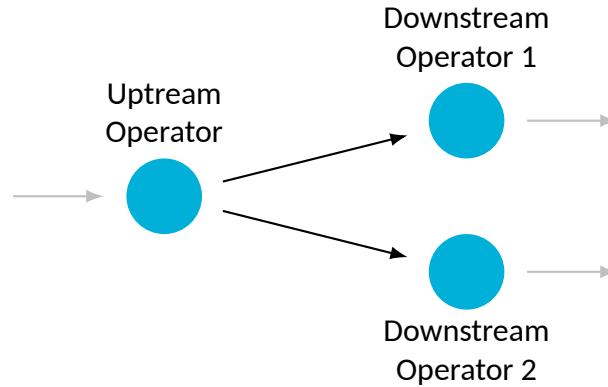


Figura 3.16: Ejemplo de una topología que cuenta con una bifurcación y su conexión entre los distintos operadores.

Fuente: Elaboración Propia.

efectiva durante la fase de Planificación del ciclo MAPE, en el proceso de asignación de operadores a los nodos móviles entrantes al sistema, restringiendo que los nodos repliquen a operadores que se conecten a dos o más operadores, y asignándolos solo a operadores que tengan una sola conexión *downstream* en la topología de procesamiento, como lo muestra la Figura 3.17.

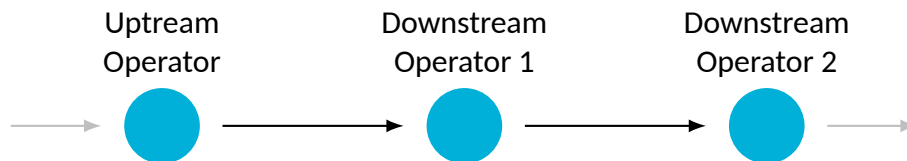


Figura 3.17: Ejemplo de una topología lineal.

Fuente: Elaboración Propia.

Gracias a esta restricción se asegura que siempre los nodos móviles se conectarán a un solo nodo, por lo que enviarán un evento a la vez. Esta restricción crea una desventaja, la cual es que se pierde el soporte hacia operadores que tiene bifurcaciones, pero esto se puede compensar con la replicación de nodos estáticos asignados a un operador que tenga dichas características, por lo que en el momento de crear la topología se debe tener en cuenta esta asignación.

3.3. INGRESO NODOS MÓVILES AL SISTEMA

Respecto al proceso de ingreso al sistema, en la fase de Monitoreo del ciclo MAPE, el actual modelo consta de dos pasos. Como se ven en la Figura 3.18 esto corresponde a un proceso constante donde el RM recibe las solicitudes de los nodos para ingresar al sistema, el cual guarda dichas solicitudes para procesarlas luego, y junto con esto le pide al nodo ingresante que envíe la información de sus recursos críticos, es decir, la intensidad de señal y la batería

restante del dispositivo. Realizando este proceso cada vez que un nodo desee ingresar al sistema independiente si es que ya lo había hecho anteriormente.

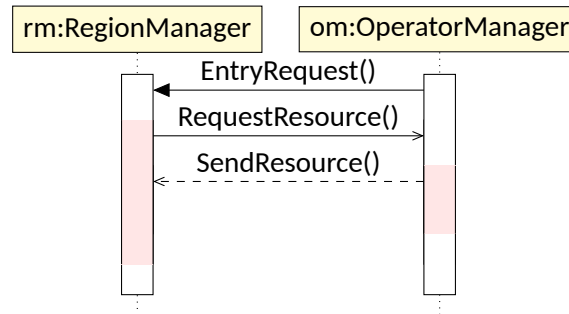


Figura 3.18: Diagrama de Secuencia del ingreso de nodos al sistema en el modelo actual.
Elaboración Propia.

Por lo que es necesario enviar tres mensajes por cada ingreso de un nodo al sistema, además esta solicitud de información es redundante, debido a que en el mismo mensaje de solicitud de ingreso al sistema el nodo ingresante puede enviar inmediatamente el estado de sus recursos.

Por lo que se cambio el procedimiento de ingreso al sistema por el envío de un solo mensaje, como se muestra en la Figura 3.19. Este mensaje cuenta con la información de los recursos críticos del nodo ingresante, es decir, su batería restante y la intensidad de señal.

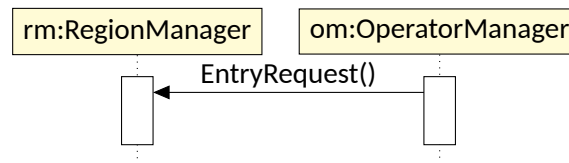


Figura 3.19: Diagrama de Secuencia del cambio al proceso de ingreso de nodos al sistema.
Elaboración Propia.

Esto reduce la cantidad de mensajes enviados a través de la ejecución del sistema al igual que una reducción en el tiempo que se demora un nodo en ingresar al sistema, pensando en la escalabilidad del modelo y como estos factores pueden afectar al desempeño del sistema.

3.4. FORMACIÓN Y MANTENCIÓN DE CLÚSTERS

Para la formación del clúster y más específicamente la mantención de este, el modelo utiliza una lista de activación que funciona como un mapa virtual de los recursos móviles del sistema, ignorando los recursos estáticos. Esta lista se crea respecto a los nodos aceptados por el RM y con la información de los recursos de cada nodo. En la Figura 3.20 se muestra un representación de esta lista, donde se ve a qué nodos están replicando a cada operador dentro del sistema.

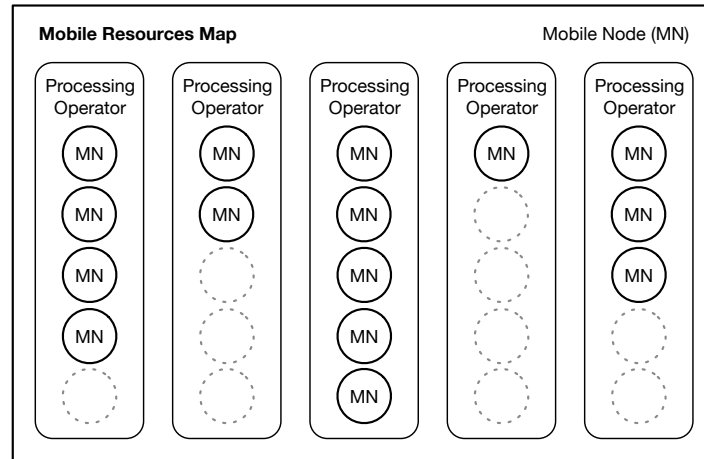


Figura 3.20: Representación gráfica del mapa virtual que posee el RM para la administración de recursos del sistema en el modelo actual.

Fuente: Elaboración Propia.

La falencia de utilizar esta lista se debe a que el RM no posee toda la información del sistema, por lo que no puede tomar las medidas necesarias en caso de algún problema se presente. Por ejemplo, en la pérdida de un nodo necesita avisar a los nodos conectados al nodo perdido que dejen de enviar eventos a dicho nodo, por tanto necesita saber cual son los nodos que se conectaba a él.

El cambio propuesto es hacer una separación entre la lista de activación y el mapa de los recursos. Por parte de la lista, esta funciona durante las fases del ciclo MAPE, la cual tendrá a los nodos que solicitaron ingresar al sistema y fueron aceptados dentro del periodo de la fase de Monitoreo. Por otro lado, el mapa tendrá la información de cuales son los nodos que están activos dentro del sistema incluyendo los nodos estáticos, junto con esto, información de cada uno de los nodos, es decir, estado de sus recursos críticos, a que nodo está conectado, ya sea *upstream* o *downstream* y a que operador esta replicando, entre otros. Una representación del este mapa se presenta en la Figura 3.21, donde se puede ver como están asignados los nodos, ya sea móvil o estático, a cada uno de los operadores y cual es la conexión entre los nodos. Cabe destacar que las líneas puntuadas y tenues corresponde a las conexiones administradas por el SPS y estas son administradas por él y no por el modelo.

Esto le permite al RM tener mayor control respecto a lo que esta sucediendo dentro del sistema, lo que también conlleva a que deba saber como están asignado los recursos de los nodos estáticos, y así poder asignar de mejor forma estos recursos y poder administrar las conexiones entre los nodos del sistema, ya sea en casos de eliminación de un nodo o la pérdida de uno de estos. También se debe tener en cuenta que este cambio podría representar que el RM sea un punto único de falla, pero debido a que se encuentra en *Nimbus* se pueden aplicar las mismas políticas preventivas que posee en SPS al RM.

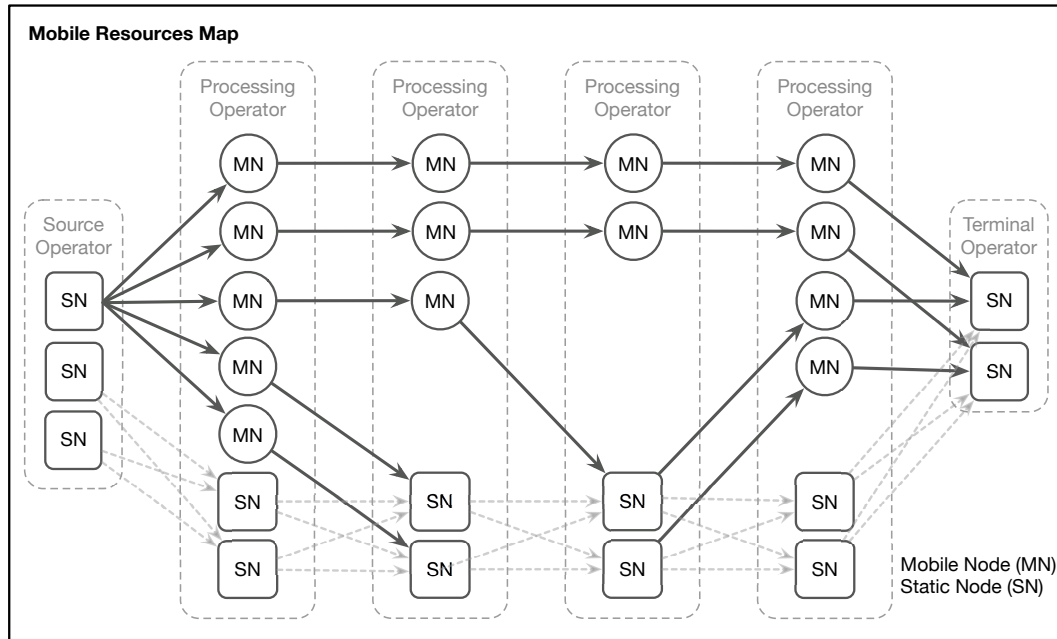


Figura 3.21: Representación gráfica del mapa virtual que posee el RM para la administración de recursos del sistema en el modelo actual.

Fuente: Elaboración Propia.

3.5. BALANCE DE CARGA HETEROGÉNEO

El actual modelo no consta de un algoritmo definido para la asignación de tareas cuando un nodo ingresa al sistema que se efectúa durante la fase de Planificación del RM. Existiendo una sola política de asignar los nodos de forma equitativa a los operadores de procesamiento de forma de distribuir la carga entre estos.

Esto no tiene en cuenta los recursos disponibles del nodo que desea ingresar al sistema, para así poder asignar de mejor forma las tareas a los nodos. Por ejemplo, asignar una tarea menos pesada a un nodo de menor capacidad y menos recursos.

Para lograr esto se asignarán las tareas de forma heterogénea, es decir, el sistema tomará en cuenta los recursos de los dispositivos e intentando de asignar los nodos a un operador que se acomode de mejor forma a sus recursos, y así aumentar el desempeño del sistema en general. Para esto, se implementó que cada vez que un nodo solicita ingresar al sistema, este analiza los recursos de dicho nodo y lo intenta de asignar a un operador bajo la política de seleccionar al operador con mayor prioridad que se adapte más a los recursos del nodo, siendo que un nodo tiene mayor prioridad si es que este es replicado por menos nodos, es decir, tiene menos poder de procesamiento asignado. Esto se traduce al Algoritmo 3.3, donde al nodo se le clasifica su tipo dependiendo de la convención de si se considera un dispositivo es de gama alta o gama baja respecto a sus recursos de hardware.

Algoritmo 3.3 Selección del Operador que se asociar a un nodo

Entrada: $tipo \leftarrow$ Tipo de nodo móvil**Salida:** Id del Operador que sera replicado por el nodo

```
1: procedure CALCULARPRIORIDAD( $op$ )
2:    $prioridad \leftarrow 0$ 
3:   for each  $nodo$  in  $op.réplicas$  do ▷ Lista de nodos que replican al operador  $op$ 
4:     if  $nodo.tipo == gama\_baja$  then
5:        $prioridad \leftarrow prioridad + prioridad\_gama\_baja$ 
6:     else if  $nodo.tipo == gama\_alta$  then
7:        $prioridad \leftarrow prioridad + prioridad\_gama\_alta$ 
8:     end if
9:   end for
10:  return  $prioridad$ 
11: end procedure

12:  $op\_min.prioridad \leftarrow M$  ▷  $M$  es un número muy alto
13: for each  $op$  in  $lista\_op$  do ▷ Lista de operadores en la topología
14:   if  $op.tarea == liviana$  then
15:      $op.prioridad \leftarrow prioridad\_tarea\_liviana + CALCULARPRIORIDAD(op)$ 
16:   else if  $op.tarea == pesada$  then
17:      $op.prioridad \leftarrow prioridad\_tarea\_pesada + CALCULARPRIORIDAD(op)$ 
18:   else
19:      $op.prioridad \leftarrow M$ 
20:   end if
21:   if  $op\_min.prioridad > op.prioridad$  then
22:      $op\_min \leftarrow op$ 
23:   else if  $op\_min.prioridad == op.prioridad$  then
24:     if  $op.tarea == liviana$  and  $tipo == gama\_baja$  then
25:        $op\_min \leftarrow op$ 
26:     else if  $op.tarea == pesada$  and  $tipo == gama\_alta$  then
27:        $op\_min \leftarrow op$ 
28:     end if
29:   end if
30: end for
31: return  $op\_min.id$ 
```

Esto va a aumentar el tiempo promedio que el nodo estará activo en el sistema y así el sistema debe enviar una menor cantidad de mensajes, debido a que reducirá el envío de mensajes de coordinación entre el RM y los OM, así disminuyendo el *overhead* del sistema y la saturación de la red, como también la disminución del uso de los recursos debido a la transmisión de estos mensajes.

3.6. UMBRAL DE BALANCE DE CARGA

Actualmente existe un umbral de sobrecarga en el modelo, el cual se encarga de detectar, durante la fase de Análisis del ciclo MAPE del OM, si es que el nodo se encuentra sobrecargado en caso de que su utilización sobre pase este umbral, notificando al nodo *upstream* de su estado para que este aplique las medidas correctivas, que en ese caso desviará el flujo hacia un nodo estático o bajará el peso de la conexión si es que aun se encuentra sobrecargado. En la Figura 3.22 se ve un ejemplo en donde en un ciclo i , el nodo móvil *downstream* se encuentra sobrecargado y el nodo *upstream* se conecta al nodo estático para dividir el flujo de eventos, siendo el peso de cada conexión de W_x . En el ciclo $i + 1$, el nodo *upstream* disminuye el peso de la conexión hacia el nodo móvil en 2α . Y por último en el ciclo n , el nodo móvil ya no se encuentra sobrecargado.

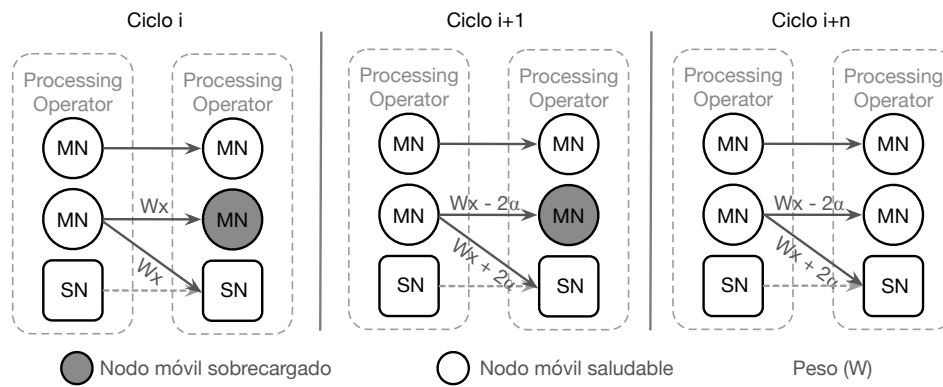


Figura 3.22: Ejemplo del proceso de liberación de carga de un nodo móvil sobrecargado.

Fuente: Elaboración Propia.

Se ha observado que los nodos al dejar de encontrarse sobrecargados bajan demasiado su utilización respecto a la que tenían anteriormente, debido a que el nodo *upstream* al recibir una notificación de que una de sus conexiones se encuentra sobrecargada agrega una conexión hacia un nodo estático lo que divide el flujo en dos, si es que el nodo a pesar de esto continua sobrecargado se sigue bajando el peso de la conexión, disminuyendo más aun el flujo que va hacia el nodo sobrecargado. Este problema se debe a que no existe un mecanismo que permita al nodo volver a un estado normal y mantener la utilización del nodo en un rango “saludable”.

Para esto se agregará un umbral inferior, el cual no permitirá que los nodos no puedan bajar de cierta utilización, asegurando una cantidad mínima de aporte al sistema. Este umbral funciona de la misma forma que el umbral superior, pero a diferencia de este, el nodo no puede sobrepasar el flujo máximo que le permite el número de conexiones actuales, es decir, si el nodo estático *upstream* posee tres conexiones hacia nodos que replican al mismo operador, el flujo máximo que puede solicitar un nodo *downstream* que se encuentra subcargado es de un tercio o 33 % del flujo total que procesa el nodo *upstream*. En la Figura 3.23 se puede ver un ejemplo de este mecanismo, donde en el ciclo i se encuentra al nodo móvil sobrecargado y el nodo *upstream* crea una conexión con un nodo estático, definiendo un peso de W_x para

cada conexión. En el ciclo $i + 1$, se disminuye la conexión al nodo móvil en 2α . En el ciclo $i + 2$, el nodo se encuentra subcargado y por último, en el ciclo $i + n$, se aumenta la conexión en α y el nodo se encuentra en un estado saludable.

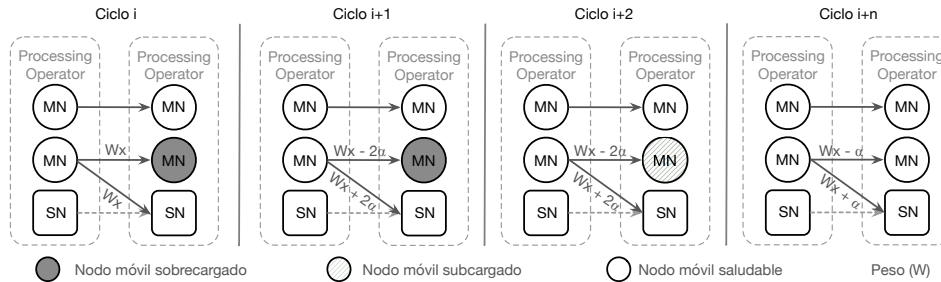


Figura 3.23: Ejemplo del proceso de regulación de la carga para un nodo móvil que se sobrecarga y luego se subcarga.

Fuente: Elaboración Propia.

Este mecanismo se encarga de mantener la utilización de los nodos dentro de un rango para poder sacar la mayor cantidad de provecho del aporte de los nodos móviles. Por otro lado afecta al *overhead* de sistema, ya que se necesita mandar más mensajes para notificar la subcarga del nodo, pero es un buen *trade-off* para el funcionamiento del sistema.

3.7. CONEXIONES ENTRE OPERATOR MANAGERS

Actualmente en el sistema, los OM se encuentran conectados entre ellos para coordinar los distintos funcionamientos del sistema, enviándose mensajes entre sí para notificar los distintos cambios necesarios para los procesos que tiene el sistema, como son los cambios en la conexión entre ellos, la sobrecarga de nodos, la eliminación de nodos del sistema, etc. Además, se le debe notificar al RM de estos cambios, pero este no cuenta con toda la información del sistema, ya que esta se encuentra dispersa a través de los nodos. Como se muestra en la Figura 3.24, cada vez que un OM debe notificar algo este debe enviar dos mensajes, uno al OM que necesita notificar y otro al RM.

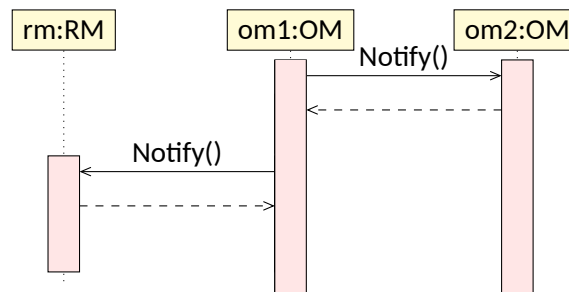


Figura 3.24: Diagrama de Secuencia del notificación entre nodos con el modelo actual.

Fuente: Elaboración Propia.

Esta conexión entre OM es redundante debido a que el RM ya se conecta con cada OM, por lo que es posible realizar esta conexión a través de él. Además, que de todas formas se debe informar de esto al RM. Junto con que se tiene que considerar que se le esta ejerciendo la carga de realizar estas notificaciones a los nodos mismo, lo que afecta a la cantidad de mensajes que envían y los recursos que utilizan para hacerlo.

Para esto se elimina la conexión entre OM y se deja encargado al RM de notificar los cambios que ocurran dentro del sistema. Es decir, que siempre que algo suceda el OM notificará al RM para que este notifique a los nodos que sean necesarios notificar. Liberando tanto de este envío de mensajes a los nodo, como manteniendo así de mejor forma la información del sistema en el RM, como se muestra en la Figura 3.25.

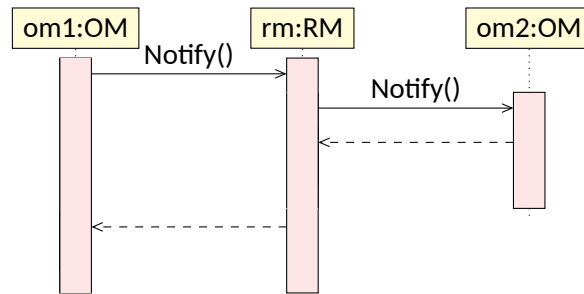


Figura 3.25: Diagrama de Secuencia del notificación entre nodos a través del RM.

Fuente: Elaboración Propia.

Esto hace que el nodo tenga que enviar un mensaje menos por cada notificación y le da más control al RM sobre que es lo que sucede sobre el sistema, así teniendo mejor información de este. También, disminuye la cantidad de información que deben tener los nodos respecto los demás, ahorrando así también recurso de los dispositivos móviles.

CAPÍTULO 4

DISEÑO EXPERIMENTAL

En este capítulo se plantean distintos experimentos para evaluar el comportamiento del sistema y de sus partes, cuales son las respuestas de este a frente a distintas situaciones y si saber sí es que su comportamiento es el esperado.

Se desarrollan distintas evaluaciones para las distintas partes del sistema, específicamente, cuatro partes. Estas son la evaluación del efecto del flujo de eventos sobre la carga de procesamiento del sistema, en donde se quiere observar el comportamiento de la utilización independiente de si es que el sistema de balance de carga entra en juego o no. La siguiente evaluación es la del efecto de la cantidad de nodos conectados sobre la carga sistema, donde de igual manera se busca observar el comportamiento de la utilización independiente de si es que el sistema de balance de carga entra en juego o no. La tercera evaluación es la del sistema de balance de carga, donde se evaluará a este sistema con respecto a distintas variables que podrían afectar su comportamiento. Por último, se evalúa el sistema de tolerancia a fallas, que también se evaluará respecto a las variables que podrían afectar a su comportamiento.

Para realizar estas evaluaciones se identifican distintas variables, las cuales se clasifican en variables de escenario y variables de decisión de diseño. Las variables de escenarios corresponden a variables que son externas al sistema, es decir, el flujo de eventos por segundos que ingresan al sistema, la topología de procesamiento, el número de nodos estáticos, el número de dispositivos móviles disponibles y el modelo de movilidad que poseen los dispositivos. Por otro lado, las variables de decisión de diseño son las variables internas al sistema, es decir, el tiempo del ciclo MAPE y el umbral de detección de sobrecarga.

Por último, estas evaluaciones se realizarán sobre ambientes simulados en *ns-3* con parámetros específicos para el consumo de energía, la latencia de procesamiento de un evento, etc., y están basadas en los resultados obtenidos del *benchmarking* realizando en [Morales, 2016]. Estos parámetros se encuentran en el Anexo A.

4.1. EFECTO DEL FLUJO DE EVENTOS SOBRE LA CARGA DE PROCESAMIENTO DEL SISTEMA

Se evaluarán cuales son los efectos que tiene la cantidad de eventos por segundo sobre la utilización promedio del sistema, es decir, como afecta a la carga de los nodos de procesamiento, si es que estos se sobrecargan o son capaces de procesar el flujo de forma normal, independiente de que el sistema de balance de carga entre en juego o no. Se espera que el flujo afecte directamente tanto a la carga de los nodos como a la latencia de los eventos.

Para ver estos efectos se realizará un experimento en el cual se aplicará un flujo variable de eventos al sistema, donde se irá aumentando este flujo desde 1 eps (evento por segundo) hasta los 200 eps en un periodo finito de una hora (3600 segundos) y se medirá la utilización general del sistema, es decir, el promedio entre la utilización de los nodos inscritos a este, donde la utilización se calcula como:

$$\text{utilización} = \frac{\text{latencia de procesamiento promedio del nodo} * \text{número de eventos que arriban}}{\text{tiempo de observación}}$$

Junto con esto se calcula la desviación estándar del este promedio, para ver entre que rangos se mueven las utilizaciones. Por otro lado, también se medirá la latencia de los eventos procesados, es decir, el tiempo que se demora dicho evento en salir desde el operador fuente hasta que llega al operador terminal.

Además, se realizará sobre una topología lineal como se observa en la Figura 4.26, con un total de 5 nodos estático asignado a cada operador. Por parte de los dispositivos móviles, habrá un máximo de 7 dispositivos, los que contarán con un modelo de movilidad ideal o estático, impidiendo que estos puedan estar fuera del rango de procesamiento y así eliminando la influencia del movimiento en los datos a medir. Por último, las variables de decisión de diseño se mantendrán solo en sus valores bases. Resumiendo, las variables para este experimento son:

1. Variables de Escenario:

- **Flujo de eventos:** Variable.
- **Topología de procesamiento:** Lineal.
- **Nodos estáticos:** 5 (1 Operador Fuente, 3 Operadores de Procesamiento, 1 Terminal).
- **Dispositivos móviles:** 7.
- **Modelo de movilidad:** Ideal/Estático.

2. Variables de Decisión de Diseño:

- **Ciclo MAPE:** Ciclo RM: 5 seg. — Ciclo OM: 6 seg.
- **Umbral de detección de sobrecarga:** Utilización > 0.6.
- **Umbral de energía:** Energía > 0.15.

4.2. EFECTOS DEL NÚMERO DE NODOS CONECTADOS SOBRE EL SISTEMA

Se quiere observar los efectos que tiene el número de nodos conectados al sistema sobre la utilización general de este, es decir, como se ven afectada la carga de los nodos y como es

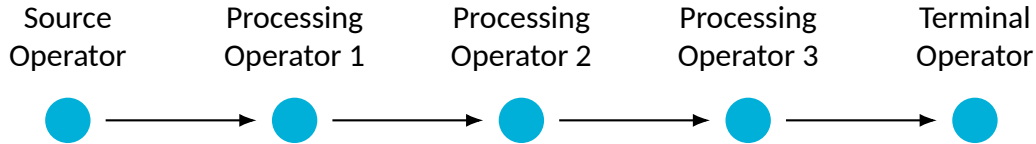


Figura 4.26: Topología lineal utilizada para realizar los distintos experimentos.

Fuente: Elaboración Propia.

que esta se va distribuyendo respecto a la cantidad de nodos que van ingresando al sistema, junto con esto, se quiere observar el comportamiento de la latencia frente a estos cambios. Se espera tener resultados que muestren que la utilización vaya bajando a medida que los nodos van ingresando al sistema, y por parte de la latencia, vaya aumentando respecto a la cantidad de nodos móviles vaya aumentando también.

Para realizar esta observación se realizará un experimento donde medirá la utilización de los nodos conectados al sistema, ya sean estáticos o móviles. Este experimento constará de tres nodos de procesamiento estáticos fijos, que replicarán a los operadores de procesamiento de la topología de la Figura 4.26, para luego ir agregando nodos móviles al sistema a medida que pasa el tiempo, siendo un total de 14 nodos los que ingresarán finalmente cada 240 segundos en un periodo total de una hora y estos nodos constarán de un modelo de movilidad ideal. Las variables definidas son:

1. Variables de escenario:

- **Flujo de eventos:** 200 eps.
- **Topología de procesamiento:** Lineal.
- **Nodos estáticos:** 5 (1 Operador Fuente, 3 Operadores de Procesamiento, 1 Terminal).
- **Dispositivos móviles:** Variable.
- **Modelo de movilidad:** Ideal/Estático.

2. Variables de decisión de diseño:

- **Ciclo MAPE:** Ciclo RM: 5 seg. — Ciclo OM: 6 seg.
- **Umbral de detección de sobrecarga:** Utilización > 0.6.
- **Umbral de energía:** Energía > 0.15.

4.3. EVALUACIÓN DEL SISTEMA DE BALANCE DE CARGA

En esta sección se evaluará el mecanismo de balance de carga, para ello se realizarán experimentos que permitirán observar el efecto que tienen sobre sistema de balance de carga las

variables que podría influir sobre su desempeño, estas son, el flujo de eventos, la topología de procesamiento, el umbral de sobrecarga y el tiempo del ciclo MAPE.

Estos experimentos consistirán en la observación de la utilización de 3 nodos móviles de un total de 7, a los cuales se le agrega una utilización adicional durante un periodo de 10 minutos de forma separada, para evaluar la reacción del sistema frente a este aumento en la carga del nodo. Se ejecutarán varias pruebas de mismo tipo, cambiando solo el valor de una de las variables mencionadas, con el objetivo de ver los efectos que dicha variable sobre el sistema. Las pruebas serán de una duración de una hora y los nodos contarán con un modelo de movilidad ideal, para impedir que los nodos salgan de la zona de procesamiento y esto influya en los datos medidos.

Los valores que tomarán las variables en las distintas pruebas son:

1. Variables de escenario:

- **Flujo de eventos:** 50 – 100 – 200 eps.
- **Topología de procesamiento:** Lineal – Bifurcada – Trifurcada
- **Nodos estáticos:** 5 – 6 (1 Operador Fuente, 3 – 4 Operadores de Procesamiento, 1 Terminal).
- **Dispositivos móviles:** 7.
- **Modelo de movilidad:** Ideal/Estático.

2. Variables de decisión de diseño:

- **Ciclo MAPE:** Ciclo RM: 3 – 5 – 7 seg. — Ciclo OM: 4 – 6 – 8 seg.
- **Umbral de detección de sobrecarga:** Utilización > 0.5 – 0.6 – 0.7.
- **Umbral de energía:** Energía > 0.15.

Las representaciones de las topologías se pueden ver en la Figura 4.26 para la topología lineal, en la Figura 4.27 para la topología bifurcada y para la topología trifurcada en la Figura 4.28. Finalmente cabe mencionar que respecto a los nodos estáticos la cantidad de estos presentes cambia respecto al tipo de topología, donde se tendrá uno por cada operador del esta.

4.4. EVALUACIÓN DEL SISTEMA DE TOLERANCIA A FALLAS

Se evaluará el desempeño del mecanismo de tolerancia a fallas, en donde se realizarán experimentos que permitan observar el efecto que tienen distintas variables sobre este mecanismo y así evaluar las repuesta de este mecanismo frente a los cambios en dichas variables.

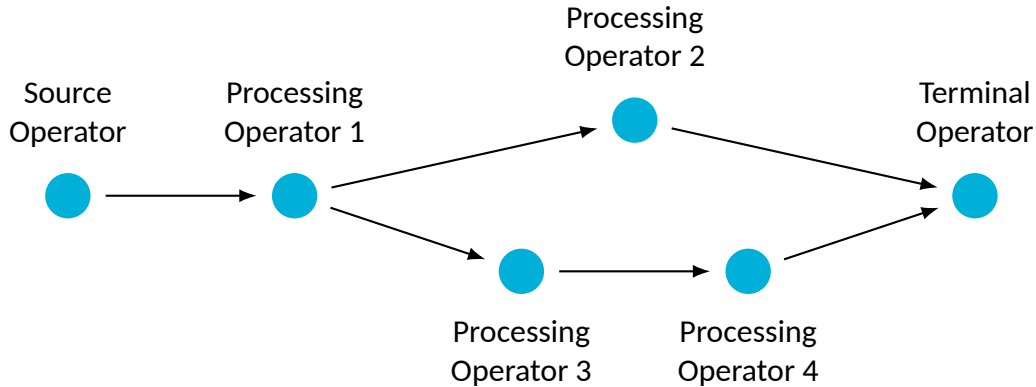


Figura 4.27: Topología bifurcada utilizada para realizar los distintos experimentos.
Fuente: Elaboración Propia.

Estas son el modelo de movilidad, el tiempo del ciclo MAPE, el umbral de energía, la Gama del Dispositivo Móvil, el flujo de Eventos y la tipología de Procesamiento.

Estos experimentos medirán la cantidad de eventos perdidos en una hora de observación, se tendrán 14 nodos móviles con un modelo de movilidad no ideal. A siete de ellos, seleccionados de forma aleatoria, se les desactivará de forma brusca para simular una falla en estos. Esto se realizará cada 450 segundos dentro de un periodo de una hora. Se debe tener en consideración que al desactivar un nodo no necesariamente va a estar activo en el sistema, por lo que se contará la cantidad de nodos que fueron efectivamente fallados, es decir, que estaban activos al momento de la falla. Debido a que los modelos de movilidad son aleatorios podrían afectar a las mediciones realizadas, por lo que se ejecutarán 10 simulaciones con distintas semillas para sacar un promedio de la cantidad de eventos perdidos entre estas simulaciones. Además, se contará la cantidad de mensajes enviados adicionales a los eventos para ver el *overhead* que posee el sistema frente a estos escenarios, y también el tiempo de recuperación del sistema.

Los valores que tomarán las variables en las distintas pruebas son:

1. Variables de Escenario:

- **Flujo de eventos:** 50 – 100 – 200 eps.
- **Topología de procesamiento:** Lineal – Bifurcada – Trifurcada
- **Nodos estáticos:** 5 – 6 (1 Operador Fuente, 3 – 4 Operadores de Procesamiento, 1 Terminal).
- **Dispositivos Móviles:** 14.
- **Modelo de Movilidad:** Gauss-Márkov – *Random Walk* – *Random Direction*.
- **Tipo de Dispositivo:** Gama Baja – Mixto – Gama Alta

2. Variables de decisión de diseño:

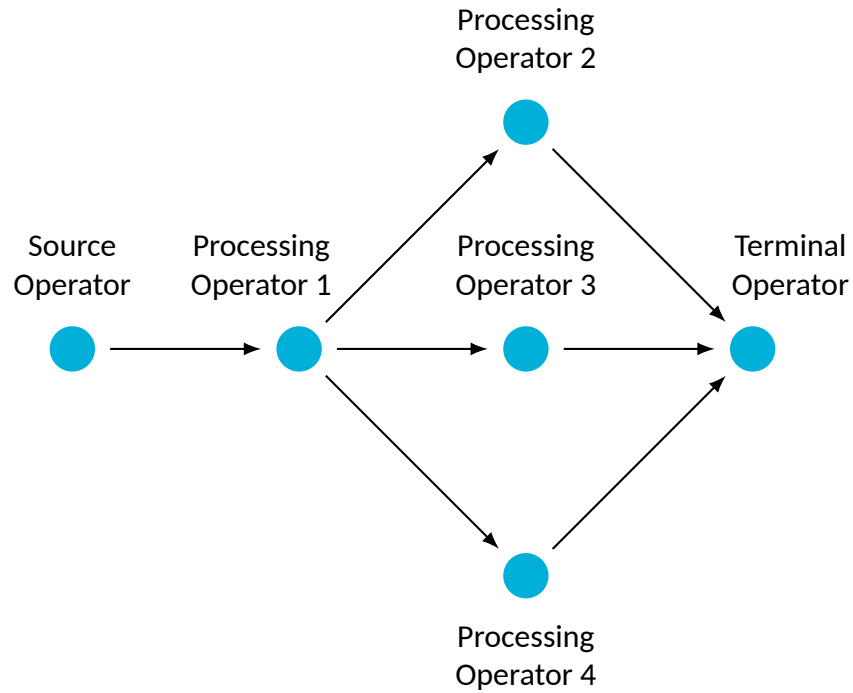


Figura 4.28: Topología trifurcada utilizada para realizar los distintos experimentos.

Fuente: Elaboración Propia.

- **Ciclo MAPE:** Ciclo RM: 3 – 5 – 7 seg. — Ciclo OM: 4 – 6 – 8 seg.
- **Umbral de detección de sobrecarga:** Utilización > 0.6.
- **Umbral de energía:** Energía > 0.05 – 0.15 – 0.25.

CAPÍTULO 5

RESULTADOS Y ANÁLISIS

En esta sección se presentan los resultados obtenidos en los experimentos presentados en Capítulo 4. Junto con esto se realizará un análisis de lo observado en dichos resultados. Estos resultados se complementan con las tablas y gráficos presentados en el Anexo B.

5.1. EFECTO DEL FLUJO DE EVENTOS SOBRE LA CARGA DE PROCESAMIENTO DEL SISTEMA

En la Figura 5.29, se observa el comportamiento del sistema, en cuanto a la utilización y latencia, frente a los cambios en el flujo de eventos. En el gráfico de la Figura 5.29a se observa como la utilización, representada por la línea verde, va subiendo junto con el aumento en el flujo de eventos, acompañada por la desviación estándar de esta, la cual es representada por la franja de color verde claro, que posee el mismo comportamiento. Estos resultados muestran que el flujo de eventos afecta directamente sobre la carga de los nodos.

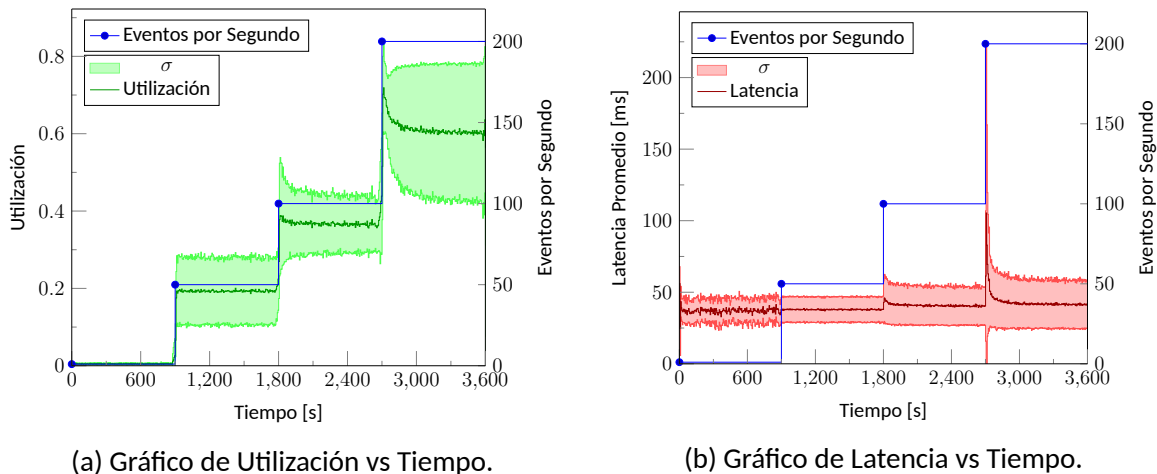


Figura 5.29: Resultados del experimento para observar los efectos del flujo de eventos sobre el sistema.

Fuente: Elaboración Propia.

Por otro lado en el gráfico de la Figura 5.29b, se observa la latencia de procesamiento de los eventos, la cual es representada por la línea roja y su desviación estándar por la franja de color más tenue. Esta muestra el comportamiento constante de la latencia, pero se observa que la desviación estándar de esta cambia y se va haciendo mayor respecto al aumento de flujo.

Junto con estos resultados se presenta los gráficos de la Figura 5.30, que muestra el compor-

tamiento de la utilización y la latencia sobre el mismo escenario de este experimento pero sin la participación del sistema. En estos se ve el mismo comportamiento de la utilización respecto a los gráficos anteriores, pero respecto a la latencia se ve un crecimiento exponencial luego de que el flujo aumenta a 200 eps.

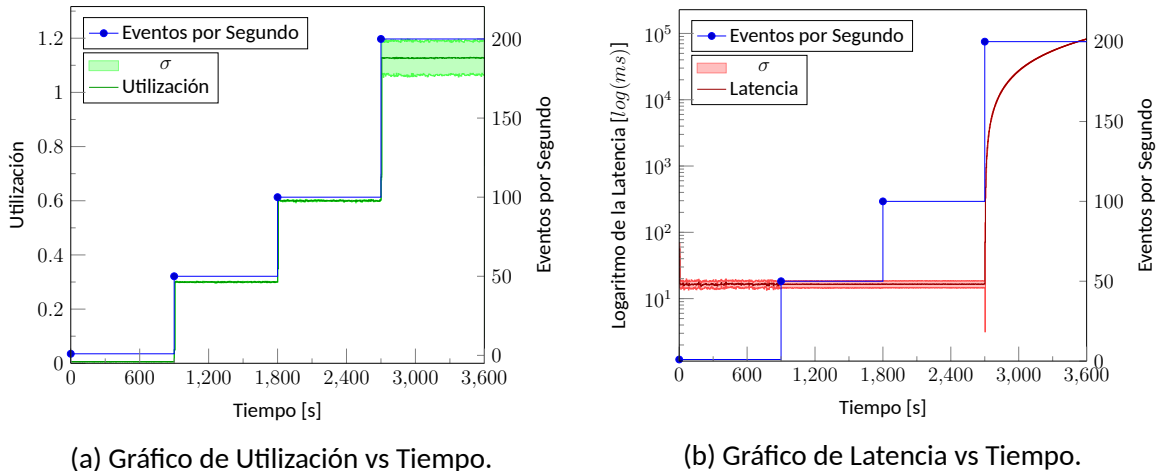


Figura 5.30: Resultados del experimento para observar los efectos del flujo de eventos sobre el sistema con solo nodo 3 nodos estáticos.

Fuente: Elaboración Propia.

De estos resultados se puede observar que los nodos sobrecargados con flujos de eventos mayores o iguales a los 200 eps, lo cual se ve reflejado en la utilización de los nodos, como se observa en el salto en la utilización en el momento en que se aumenta el flujo de 100 a 200 eps. También se ve reflejado en la latencia de los eventos de esta misma manera, con un aumento radical en el tiempo que se demoran los eventos en ser procesados al momento de aumentar el flujo. Estos efectos se ven más claramente en los gráficos de la Figura 5.30, los cuales no poseen nodos móviles.

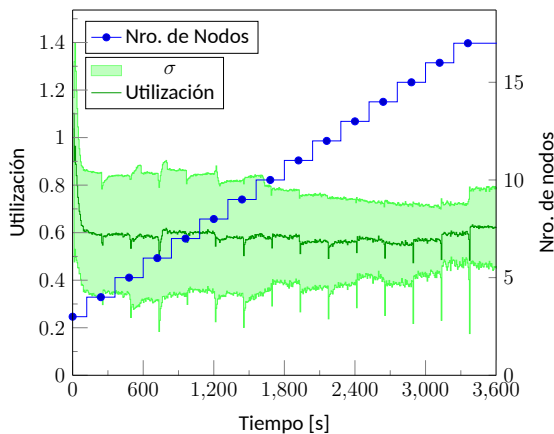
Al comparar los gráficos del sistema con y sin nodos móviles, se puede ver el aporte de los nodos móviles al sistema y como estos ayudan a bajar la utilización de los nodos estáticos y también a mantener la latencia de los eventos constante, además, cuando los nodos se encuentran sobrecargados debido al flujo de 200 eps, el sistema de balance de carga distribuye la carga entre los nodos de forma que ningún nodo sobrepase un factor de 0.8 de utilización y como se puede ver en la desviación estándar, esta se hace mayor debido a que hay más variedad dentro de los valores de utilización de cada nodo, donde los nodos con mayor poder de procesamiento se llevan la mayor parte de la carga y los con menor poder se llevan menos parte de esta.

Esto mismo afecta a la latencia debido a que los eventos pasan a ser procesados por nodos con menor poder de procesamiento por tanto se demoran más en ser procesados. En el caso de no poseer nodos móviles, como se puede ver en el gráfico de la Figura 5.30b, la latencia va creciendo exponencialmente cuando los nodos estáticos deben procesar un flujo de 200 eps, esto se debe a que las colas de procesamiento han colapsado y el tiempo que los eventos

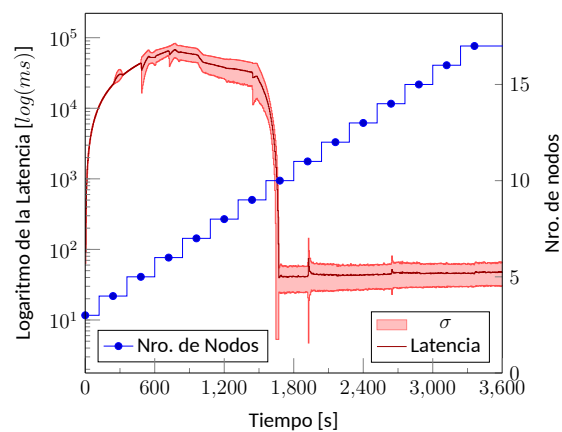
deben esperar para ser procesados por un nodo es cada vez mayor.

5.2. EFECTOS DEL NÚMERO DE NODOS CONECTADOS SOBRE EL SISTEMA

En los resultados presentados en la Figura 5.31, se observa el comportamiento de la utilización de los nodos y la latencia de los eventos cuando van ingresando nodos móviles al sistema de forma periódica, los cuales deben procesar un flujo, que sobrecarga al sistema sin nodos móviles, de 200 eps. En el gráfico de utilización se ve como esta va disminuyendo rápidamente a medida que ingresan los nodos móviles al sistema para luego mantenerse constante durante el resto del tiempo observado, fluctuando alrededor del umbral de sobrecarga de 0.6. La desviación estándar, representada por la franja de color verde claro, muestra como se distribuye los valores de la utilización entre los nodos, donde la desviación va disminuyendo a medida que ingresan más nodos al sistema. Estos resultados difieren de lo esperado, lo que correspondía a una recta con pendiente negativa, donde la utilización bajaría a medida que iban ingresando los nodos al sistema, de manera constante. Esta proposición es en un principio verdadero, ya que la utilización si baja a medida que ingresan los nodos móviles al sistema, pero cuando el sistema ya se estabiliza, es decir, que los nodos ya no se encuentran sobrecargados por este flujo, la utilización general de este se mantiene constante. Esto tiene que ver, como se vio en los resultados anteriores, con que la carga de procesamiento que se ejerce sobre el sistema no cambia, y este se encarga solo de repartirla entre los nodos que se encuentra en él.



(a) Gráfico de Utilización vs Tiempo.



(b) Gráfico de Latencia vs Tiempo.

Figura 5.31: Resultados del experimento para observar los efectos del flujo de eventos sobre el sistema con solo 3 nodos estáticos.

Fuente: Elaboración Propia.

Por parte de la latencia se puede ver como esta sube rápidamente de forma exponencial y a medida que ingresan más nodos al sistema esta baja de la misma forma para luego mantenerse constante al rededor de los 40 [ms] durante el resto del tiempo de observación.

La desviación estándar se mantiene uniforme luego de que se estabiliza la latencia de los eventos, y esta fluctúa entre el rango de las mediciones de latencia. Este resultado se esperaba que fuera de esta manera, pero cabe destacar el comportamiento exponencial con que aumenta la latencia en el principio de la observación.

Analizando estos resultados se puede ver el comportamiento del sistema a medida que va balanceando la carga entre los nodos, donde los primeros nodos se encuentran sobrecargados debido al alto flujo de eventos y a medida que van ingresando rápidamente los demás nodos al sistema, estos van pasando a un estado saludable hasta llegar a un nivel constante. El balance de carga se puede ver tanto en el gráfico de utilización en la Figura 5.31a como en el gráfico de latencia en la Figura 5.31b, donde en el primero se observa que la desviación estándar va disminuyendo, por lo que los valores de utilización se van haciendo cada vez más homogéneos dentro de los nodos, pero estos se ven afectados por el umbral de sobrecarga que no les permite que suban más allá de los 0.6, resguardando el uso imprudente de recursos móviles pero permitiendo el soporte a los nodos estáticos que de otra manera se verían sobrecargados. Finalmente, por parte de la latencia, en un principio se puede apreciar que los nodos se encuentran totalmente sobrecargados, pero rápidamente el sistema es capaz de distribuir la carga para que vuelvan a un estado saludable lo que reafirma la idea de que el sistema puede balancear la carga.

5.3. EVALUACIÓN DEL SISTEMA DE BALANCE DE CARGA

En esta sección se presentarán los resultados de los experimentos realizados para la evaluación del sistema de balance de carga. Esta consta de cuatro partes, las que corresponden a las evaluaciones de los efectos de las variables a las que está suscrito este sistema, es decir, el flujo de eventos, la topología de procesamiento, el umbral de sobrecarga y el tiempo del ciclo MAPE.

5.3.1. EFECTO DEL FLUJO DE EVENTOS SOBRE EL BALANCE DE CARGA

En los gráficos presentados en la Figura 5.32 se muestra el comportamiento de los nodos observados frente a distintos flujos de eventos. Cada línea de color representa la utilización de uno de los 3 nodos observados dentro de este experimento. A estos nodos se les agrega una utilización adicional de 0.2 en sus mediciones para que se sobrecarguen. Esta sobrecarga se realiza de forma separada para cada nodo, la que consta de un tiempo de 10 minutos y se realiza en los 15 min. de simulación para el primer nodo, a los 30 para el segundo y a los 45 para el tercero. Estas pruebas contaron con una topología lineal, el umbral de sobrecarga fijado en 0.6 y un tiempo del ciclo MAPE de 5 seg. para el RM.

Como es de esperar para los 50 eps, como se muestra en la Figura 5.32a, no se ve mayor sobrecarga para los nodos a pesar de agregar la utilización adicional a ellos, siendo el nodo

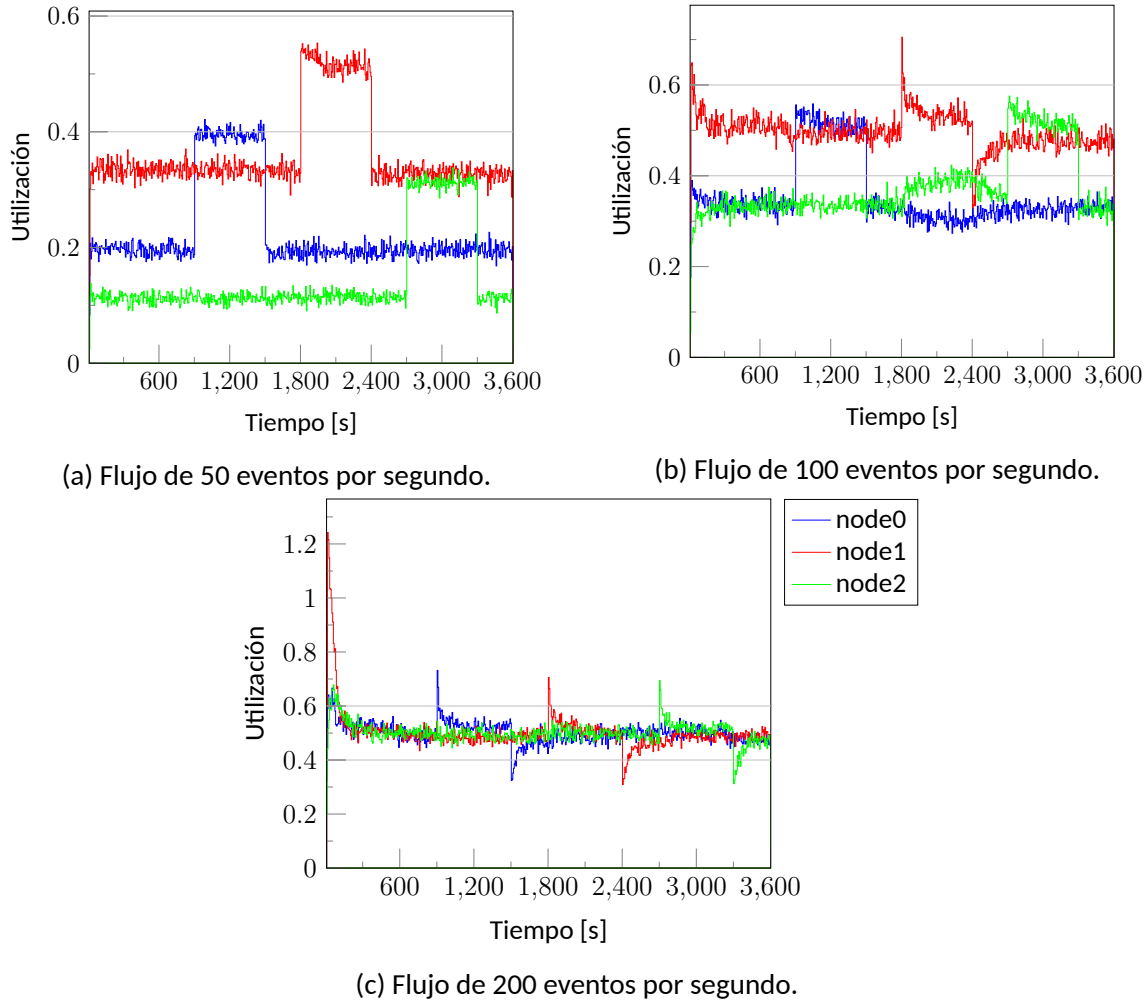


Figura 5.32: Gráficos de Utilización vs Tiempo de los resultados de los experimento para observar los efectos del flujo de eventos sobre el balance de carga.

Fuente: Elaboración Propia.

con más carga el nodo *node1*, por lo que el sistema de balance no debe entrar en funcionamiento, pues no se sobrepasa el umbral del 0.6 de utilización.

Para el caso que se muestra en la Figura 5.32b que consta de 100 eps, se observa que el nodo *node1* se encuentra sobrecargado en el comienzo de la simulación con una utilización por sobre 0.6 para luego mantenerse dentro del rango ideal entre los 0.6 y los 0.4. Al agregar la utilización adicional se ve que este se sobrecarga nuevamente pero de forma inmediata empieza a bajar su utilización. Esto indica un buen funcionamiento del sistema de balance de carga, ya que rápidamente este se ha percatado de dicha sobrecarga y ha podido balancear la carga del nodo entre los demás y así impidiendo que el nodo *node1* se vea sobrecargado por mucho tiempo. Más específicamente, el *node1* replica a *Processing Operator* 1 de la topología lineal que se encuentra en la Figura 4.26 por lo que su nodo *upstream* es el un nodo estático que replica a el *Source Operator* y al verse sobrecargado *node1* este nodo

estático desvía flujo hacia el resto de los nodos estáticos. Luego de los 10 minutos de tener la utilización adicional se puede observar que la utilización normal del nodo se encuentra bajo el umbral inferior de 0.4 por lo que el sistema aplica las medidas correspondientes y aumenta la carga del nodo para que vuelva a encontrarse dentro del rango ideal. En comparación, debido a que los otros dos nodos nunca se ven sobrecargados, su flujo es siempre el máximo posible por lo que umbral inferior no tiene efecto sobre ellos.

En el caso de los 200 eps, como se ve en la Figura 5.32c, se ve que el comportamiento de los tres nodos observados es el mismo que del nodo *node1* en el caso de los 100 eps, esto se explica por que debido al aumento de flujo los nodos se ven con mayor carga por lo que es más fácil que se encuentren sobrecargados.

Cabe destacar respecto al *overhead*, la cantidad de mensajes de *OVERLOAD* que se han enviado respecto a cada flujo como se muestra en la Tabla 5.1, donde a contraposición de lo esperado esta cantidad se hace menor a medida que mayor es el flujo. Esto se explica debido al umbral inferior o umbral de subcarga, ya que los nodos al encontrarse subcargados envían mensajes para poder tener una mayor cantidad de utilización, por lo que a tener un mayor flujo de datos y por ende tener una mayor utilización no se encuentran subcargados. Por otro lado, un nodo al encontrarse sobrecargado envía estos mensajes hasta que se encuentra dentro del rango ideal y por ende no envía más mensajes hasta que sobrepase alguno de estos umbrales, pero esto va depende de cambios en el ambiente que se esta ejecutando el sistema y no es algo que cambie fácilmente.

Tabla 5.1: Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del flujo de eventos sobre el balance de carga.

Fuente: Elaboración Propia

	50 eps	100 eps	200 eps
<i>CHECKPOINTING</i>	1058	1058	1058
<i>OVERLOAD</i>	1618	1083	197
<i>DELETE</i>	0	0	0
<i>UPDATE</i>	1064	1064	1064

5.3.2. EFECTOS DEL TIPO DE GRAFO DE PROCESAMIENTO SOBRE EL BALANCE DE CARGA

Los resultados presentados en la Figura 5.33, muestran el comportamiento de la utilización de los tres nodos observados frente a distintas topologías de procesamiento. Estas pruebas fueron realizadas de la misma manera que las de la sección anterior, pero con la diferencia que estas tuvieron un flujo de 100 eps, un umbral de sobrecarga de 0.6 y un tiempo del ciclo MAPE de 5 seg. para el RM.

En estos resultados se puede ver que la carga que poseerá un nodo dependerá en que posición se encuentra dentro de la topología, si se observa la carga que posee el nodo *node1*

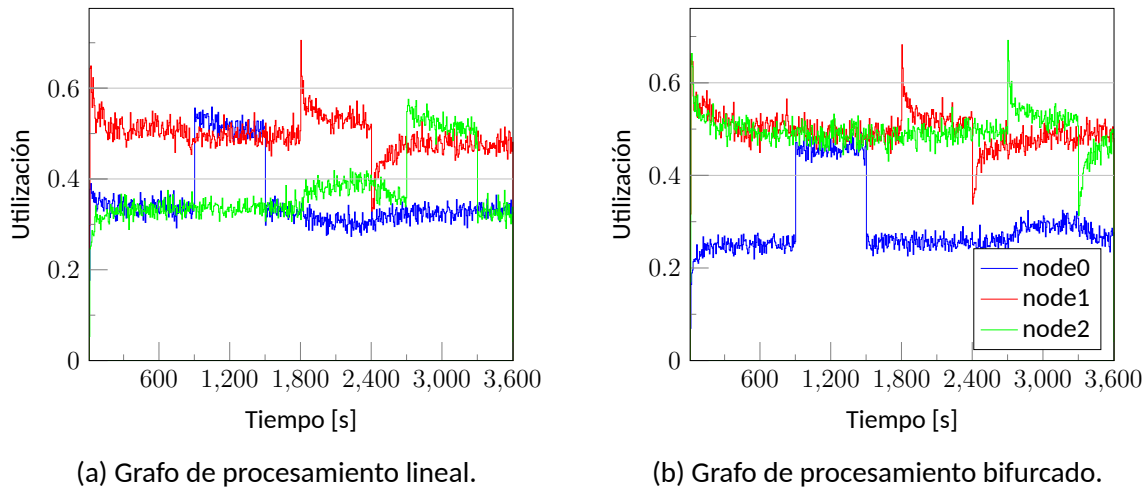


Figura 5.33: Gráficos de Utilización vs Tiempo de los resultados de los experimento para observar los efectos del tipo de grafo de procesamiento sobre el balance de carga.

Fuente: Elaboración Propia.

en el gráfico de la Figura 5.33a, esta se traslada también al nodo *node2* cuando hay una topología bifurcada como se muestra en la Figura 5.33b, esto es debido a que esta bifurcación hace que el mismo flujo vaya hacia ambos nodos de igual manera, por lo que presentan el mismo comportamiento de sobrecarga cuando a estos se les agrega la utilización adicional. Específicamente, el nodo *node1* replica al operador *Processing Operator 2* y el nodo *node2* replica al operador *Processing Operator 3*. A diferencia de esto, el nodo *node0* tiene una carga mucho más baja comparada con los otros dos nodos, debido a que este nodo móvil replica al operador *Processing Operator 4* y está conectado a un nodo móvil que emite un menor flujo de eventos a diferencia de un nodo estático que emite un mayor flujo debido a su capacidad de procesamiento.

5.3.3. EFECTOS DEL UMBRAL DE SOBRECARGA SOBRE EL BALANCE DE CARGA

Los gráficos presentados en la Figura 5.34 muestran el comportamiento de la utilización respecto a los distintos umbrales de sobrecarga. Estas pruebas fueron realizadas con un flujo de eventos de 200 eps, un grafo de procesamiento lineal y por último, un tiempo de ciclo MAPE de 5 seg. Y se realizaron de la misma forma que las anteriores, con la observación de la utilización de los tres nodos y la sobrecarga de estos en distintos periodos de la observación.

Se puede ver un comportamiento similar dentro de estos, donde la utilización se mantiene dentro del rango ideal. Estos resultados son de esperarse ya que tanto el mecanismo de sobrecarga como el de subcarga funcionan de manera correcta.

Junto con estos gráficos se presenta la Tabla 5.2, donde se ve la cantidad de mensajes adicionales que se enviaron durante el experimento, siendo el número correspondiente a *OVER-*

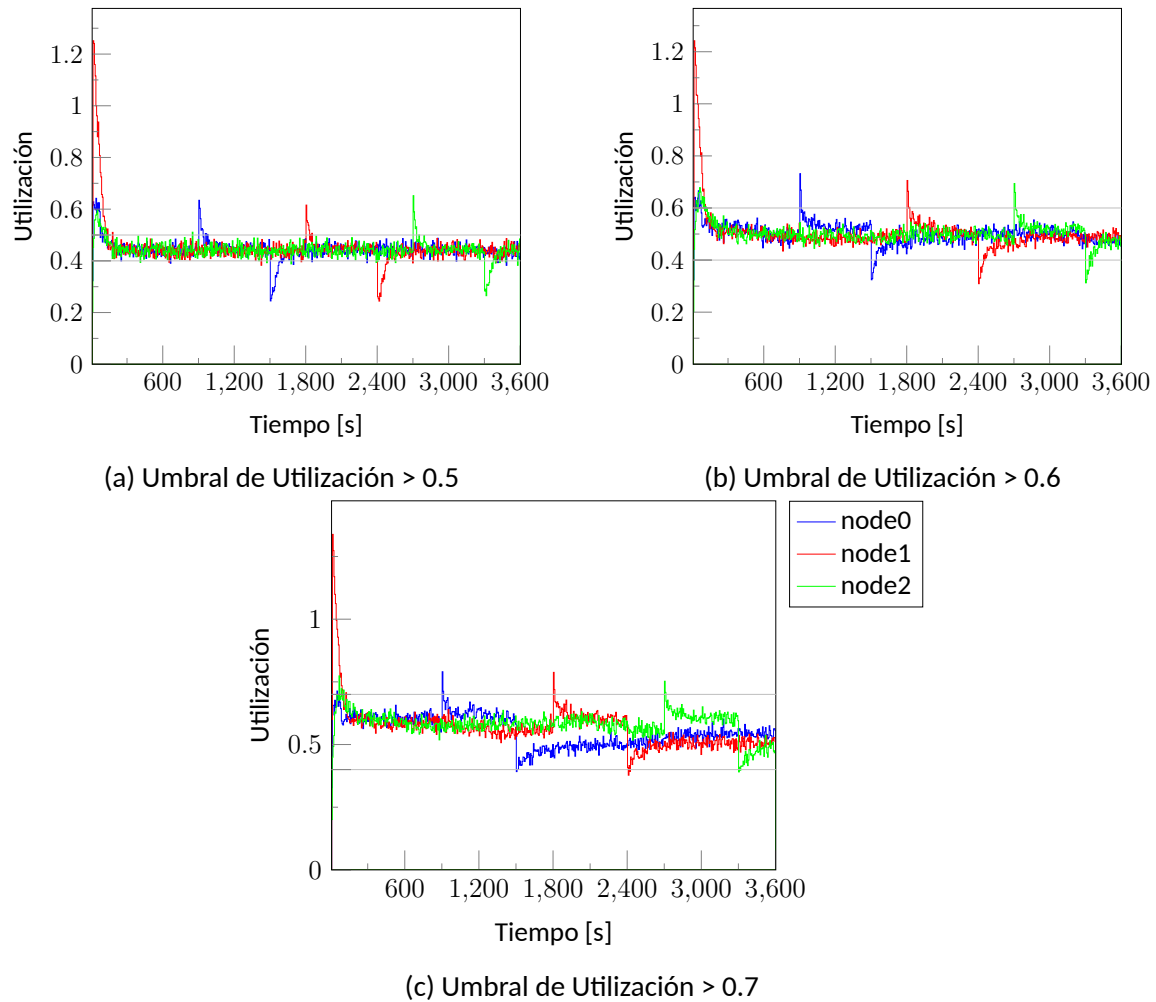


Figura 5.34: Gráficos de Utilización vs Tiempo de los resultados de los experimento para observar los efectos del umbral de sobrecarga sobre el balance de carga.

Fuente: Elaboración Propia.

LOAD la cantidad de mensajes de sobrecarga. En la tabla se aprecia de que a medida que el umbral es mayor, menor es la cantidad de mensajes de sobrecarga enviados, esto se debe a que se le da mayor libertad a los nodos para que su utilización varíe, en contraposición de el mayor provecho de la utilización. Esto ultimo, es una decisión que debe tomar en el momento de implementar el modelo, debido a que se debe sopesar que es lo que más se necesita para el caso en particular, es decir, si es que se valora más que los nodos aporte la mayor cantidad de poder de procesamiento o el tráfico de la red debido a la cantidad de mensajes que se enviarán.

Tabla 5.2: Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto de el umbral de sobrecarga sobre el balance de carga.

Fuente: Elaboración Propia

	Utilización > 0.5	Utilización > 0.6	Utilización > 0.7
<i>CHECKPOINTING</i>	1058	1058	1058
<i>OVERLOAD</i>	476	98	46
<i>DELETE</i>	0	0	0
<i>UPDATE</i>	1064	1064	1064

5.3.4. EFECTO DEL TIEMPO DEL CICLO MAPE SOBRE EL BALANCE DE CARGA

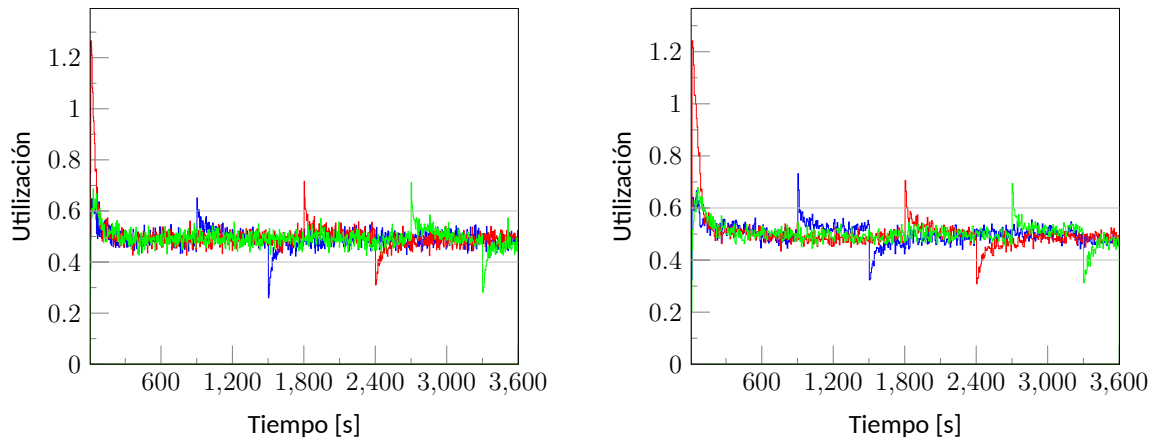
En los resultados presentados en los gráficos de la Figura 5.35 son de las pruebas realizadas para ver los efectos del tiempo del ciclo MAPE sobre el balance de carga. Estas fueron realizadas de igual manera que las pruebas presentadas anteriormente, pero con la diferencia de que contaron con un flujo de 200 eps, una topología de procesamiento lineal y un umbral de procesamiento de 0.6.

En estos gráficos se puede ver que el comportamiento de estos no varia mucho, pero si varia la cantidad o espaciado de la utilización en el tiempo, lo que era de esperar, pero se ve que estos nodos pueden volver a estar dentro de un estado saludable de forma más rápida si es que el ciclo es de menor tiempo. En la Figura 5.36 se muestran gráficos de la latencia de los eventos para ver si es que hay algún efecto sobre ella, pero no se aprecia nada significativo. Por parte del *overhead*, que puede ver en la Tabla 5.3, como era de esperarse se envían mayor cantidad de mensajes a menor tiempo del ciclo, esto se debe a que hay una mayor cantidad de ciclos por unidad de tiempo y dependiente de que tipo de mensajes sea la respuesta del sistema va ser más rápida, por lo que hay mayor trafico dentro del sistema si es que el tiempo de ciclo es menor, que no necesariamente habla de un mejor desempeño como se ve en estos resultados, y al igual que en la sección anterior, es necesario sopesar que es lo que más se acomoda a la implementación del sistema.

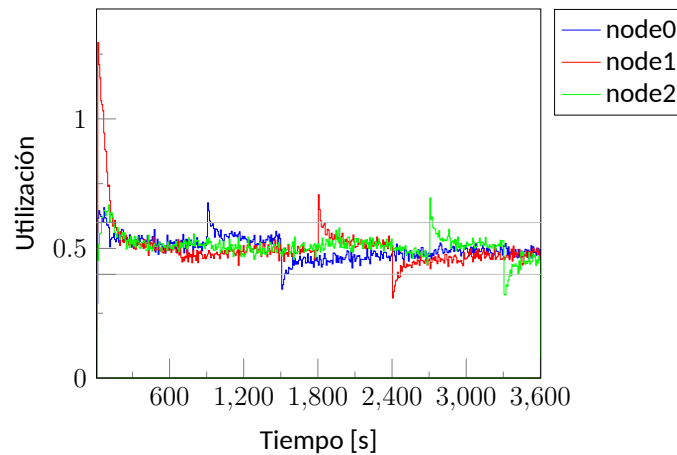
Tabla 5.3: Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del tiempo del ciclo MAPE sobre el balance de carga.

Fuente: Elaboración Propia

	RM = 3 seg. OM = 4 seg.	RM = 5 seg. OM = 6 seg.	RM = 7 seg. OM = 8 seg.
<i>CHECKPOINTING</i>	1588	1058	794
<i>OVERLOAD</i>	197	98	74
<i>DELETE</i>	0	0	0
<i>UPDATE</i>	1593	1064	799



(a) Tiempo de Ciclo MAPE RM = 3 seg. y OM = 4 seg. (b) Tiempo de Ciclo MAPE RM = 5 seg. y OM = 6 seg.



(c) Tiempo de Ciclo MAPE RM = 7 seg. y OM = 8 seg.

Figura 5.35: Gráficos de Utilización vs Tiempo de los resultados de los experimento para observar los efectos del tiempo del Ciclo MAPE sobre el balance de carga.

Fuente: Elaboración Propia.

5.3.5. ANÁLISIS FINAL PARA EL BALANCE DE CARGA

Finalmente, de lo observado se puede aseverar que el sistema de balance de carga realiza su función dentro del sistema, es decir, puede disminuir la carga de uno si es que este se encuentra por sobre los límites definidos y distribuye esta carga liberada dentro de los demás nodos. Esto permite que las colas de procesamiento de los nodos no se vean colapsadas, teniendo en cuenta de que el flujo de eventos que procesa el sistema es uno el cual puede ser procesado con las capacidades que cuenta el sistema en ese momento, por ejemplo, si es que ni los nodos estáticos son capaces de procesar 1000 es difícil que el apoyo de los nodos móviles sea significativo, pero si es que es el caso de un aumento de 100 a 200 eps, como se ha mostrado dentro de estos resultados, el soporte del modelo es significativamente positivo permitiendo que este sistema no colapse.

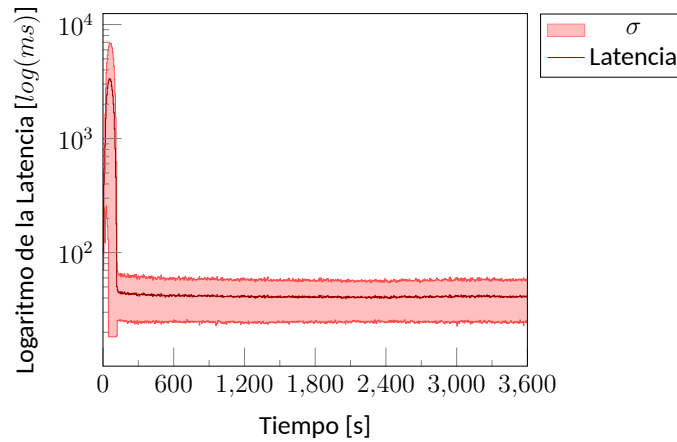
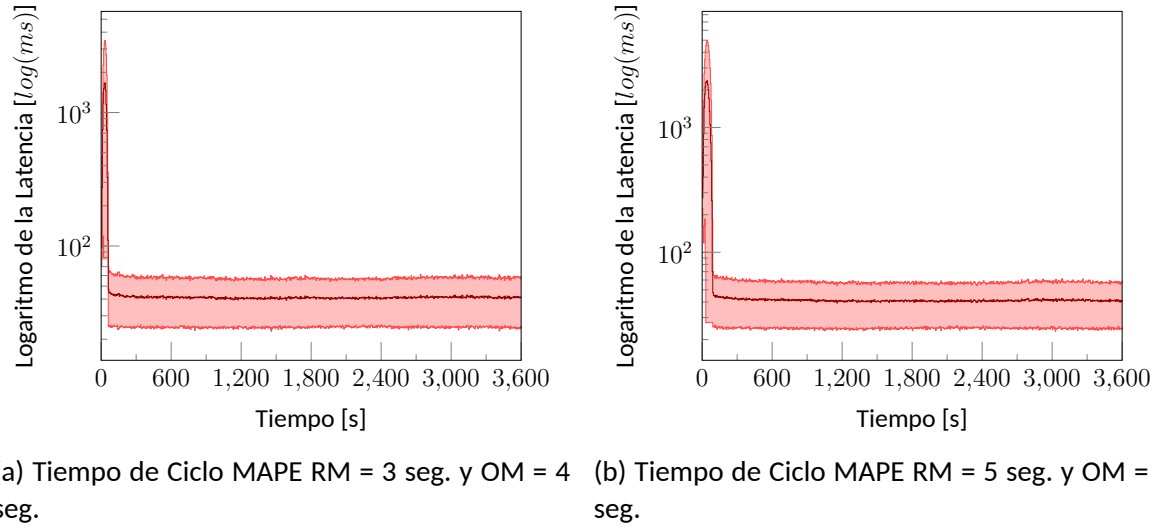


Figura 5.36: Gráficos de Latencia vs Tiempo de los resultados de los experimento para observar los efectos del tiempo del Ciclo MAPE sobre el balance de carga.

Fuente: Elaboración Propia.

5.4. EVALUACIÓN DEL SISTEMA DE TOLERANCIA A FALLAS

En esta sección se presentarán los resultados de los experimentos realizados para la evaluación del sistema de tolerancia a fallas. Esta consta de seis partes, las que corresponden a la evaluación de los efectos de las variables, a las que está suscrito el sistema de tolerancia a fallas: el modelo de movilidad, el tiempo del ciclo MAPE, el umbral de energía, la gama del dispositivo móvil, el flujo de eventos y la topología de procesamiento.

5.4.1. EFECTO DEL MODELO DE MOVILIDAD SOBRE LA TOLERANCIA A FALLAS

Los resultados presentados en la Tabla 5.4 muestran el promedio obtenido de las diez simulaciones realizadas por cada tipo de modelo de movilidad que pueden tener los nodos móviles. Se pueden observar las diferencias en los datos medidos, destacando inmediatamente la diferencia en la cantidad de eventos perdidos que fue medida en las simulaciones que se realizaron con el modelo de movilidad de Gauss-Márkov. Esta cantidad promedio de eventos perdidos es mayor a los otros dos promedios, independiente del número de nodos efectivamente fallidos, que en este caso es de un promedio de 2 nodos por simulación. Aun más, se ve que la desviación estándar de la cantidad de eventos perdidos es alrededor de cinco veces mayor que las otras desviaciones calculadas.

Tabla 5.4: Tabla resumen de los datos medidos en las pruebas para evaluar el efecto del modelo de movilidad sobre la tolerancia a fallas.

Fuente: Elaboración Propia

	Gauss-Márkov		Random Walk		Random Direction	
	Promedio	σ	Promedio	σ	Promedio	σ
Tiempo de recuperación [s]	7.25	3.58	8.69	3.23	8.6	3.40
Número de nodos fallidos	2.00	0.82	2.60	1.26	2.10	1.10
CHECKPOINTING	4122.10	1630.68	4350.90	1117.34	3904.60	1103.58
OVERLOAD	6343.20	827.74	6074.80	631.86	5624.10	799.14
DELETE	1762.70	473.73	1335.00	503.64	1102.60	407.93
UPDATE	11514.90	959.87	11070.40	721.16	11025.70	654.38
Eventos perdidos	975.50	2127.31	588.90	372.71	488.40	420.35

Analizando el total de mensajes enviados, aparte de los eventos, se puede ver en el promedio que se envía con el modelo de Gauss-Márkov es de 23.742,9 mensajes, *Random Walk* es de 23.742,9 y *Random Direction*, 21.657, por lo que existe una correlación entre estos valores y la cantidad de eventos perdidos, pero no se explica la desviación que tiene Gauss-Márkov. Al revisar los datos de las simulaciones, se encontró que existe un *outlier* en los valores medidos de 7.019 eventos perdidos. Al obtener nuevamente el promedio de los datos, obviando al *outlier*, se tiene un promedio de 304 eventos perdidos y una desviación estándar de 135.57, por lo que esto se debe tener en cuenta para el resto de los análisis, ya que se utilizó este modelo como base para la realización del resto de experimentos debido al tipo de movimiento que genera. Por parte de los otros valores tomados no se ve una mayor desviación de estos valores.

Si se tiene en cuenta el tipo de movimiento que genera cada uno los modelos, se puede apreciar que existe una correlación entre estos y la cantidad de mensajes de *DELETE* enviados, es decir, mensajes de desactivación, ya sea para la eliminación de una conexión o, derechamente, mensajes de aviso a nodo de su desactivación. Gauss-Márkov, que genera un mayor movimiento a través del área de observación, como se ve en la Figura 5.37, posee mayor cantidad de estos mensajes. Esto se debe a que en este modelo es más probable que el nodo entre y salga de la zona de procesamiento, así también para *Random Walk*, pero este

tiende a quedarse dentro de determinado espacio. Y finalmente, *Random Direction*, tiende a realizar viajes dentro del área de extremo a extremo, que permite al nodo estar dentro de la zona de procesamiento durante periodos más extensos, disminuyendo así la cantidad de mensajes de eliminación que se le envían a los nodos.

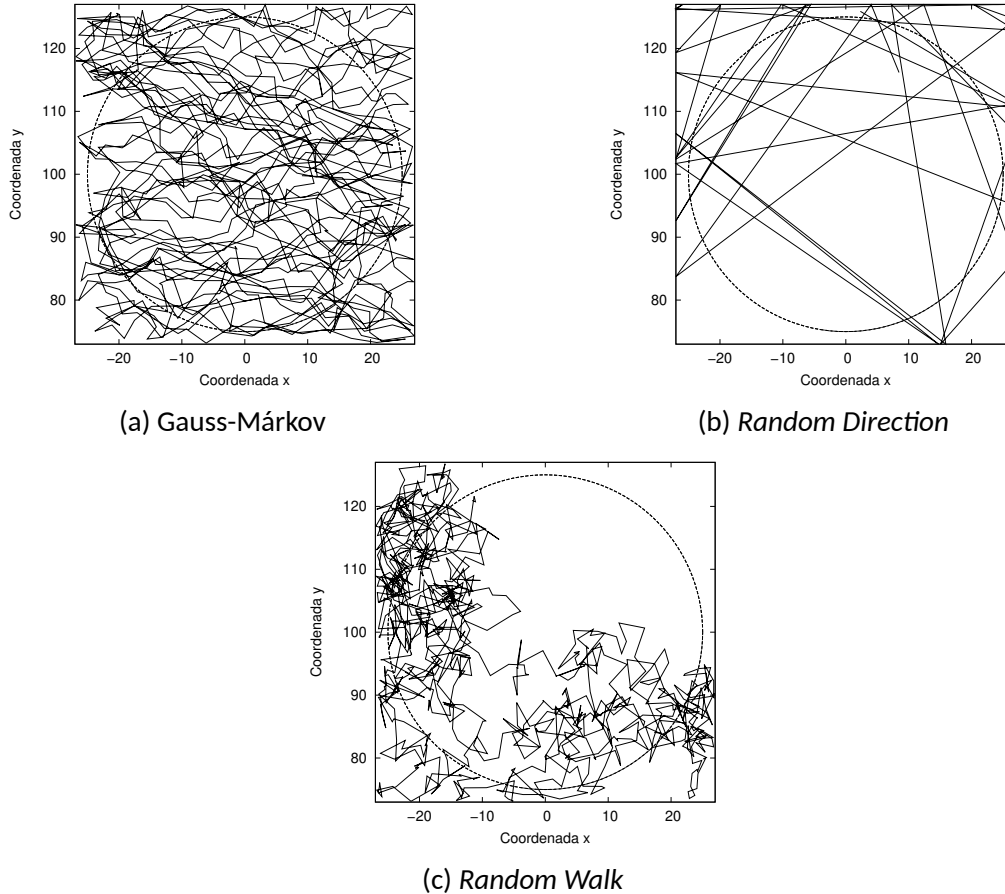


Figura 5.37: Diagramas del movimiento que presentan los nodos con los distintos modelos de movilidad.

Fuente: [Morales, 2016].

5.4.2. EFECTO DEL TIEMPO DEL CICLO MAPE SOBRE LA TOLERANCIA A FALLAS

En los resultados que se presentan en la Tabla 5.5, como es de esperar, se ve que hay un aumento en la cantidad de mensajes enviados entre los nodos a menor tiempo de ciclo. Con 33191.5 mensajes enviados cuando se cuenta con un ciclo RM de 3 seg, 23742.9 mensajes enviados cuando se cuenta con un ciclo RM de 5 seg. y 18779.3 cuando se cuenta con uno de 7 seg. Las mediciones obtenidas para el ciclo RM de 3 seg. experimentan un aumento de un 139,80 % respecto al ciclo de 5 seg. y 176,74 % con respecto al ciclo de 7 seg. Lo que habla de que se tiene una relación indirectamente proporcional, que a medida que aumenta el tiempo de los ciclos menor es la cantidad de mensajes enviados.

Básicamente, esto se debe a que hay más ciclos dentro del tiempo observado, por tanto los nodos envían mensajes de forma más seguida, aumentando el número de mensajes que viaja por la red en un determinado tiempo.

Tabla 5.5: Tabla resumen de los datos medidos en las pruebas para evaluar el efecto del tiempo del ciclo MAPE sobre la tolerancia a fallas.

Fuente: Elaboración Propia

	RM = 3 seg. OM = 4 seg.		RM = 5 seg. OM = 6 seg.		RM = 7 seg. OM = 8 seg.	
	Promedio	σ	Promedio	σ	Promedio	σ
Tiempo de recuperación [s]	10.18	2.81	7.25	3.58	7.00	3.84
Número de nodos fallidos	1.70	0.82	2.00	0.82	2.00	1.25
CHECKPOINTING	5999.90	2602.10	4122.10	1630.68	3236.10	1217.33
OVERLOAD	8578.90	734.28	6343.20	827.74	4882.00	795.59
DELETE	1815.10	505.29	1762.70	473.73	1662.40	476.09
UPDATE	16797.60	1025.24	11514.90	959.87	8998.80	803.58
Eventos perdidos	421.44	194.00	304.00	135.57	467.22	315.41

Por otro lado, se puede ver el significativo aumento en el tiempo de recuperación que poseen las pruebas realizadas con un tiempo de ciclo RM de 3 seg. y de OM de 4 seg. en comparación con los otros dos tiempos. Este aumento es de aproximadamente de un 40 %. Y aunque podría ser intuitivo pensar que, al tener un menor tiempo de ciclo, le permitiría al sistema darse cuenta de forma más rápida si es que un nodo ha fallado. Hay que tener en cuenta que con un menor tiempo del ciclo hay una mayor cantidad de mensajes enviados, como ya se menciono, lo que lleva a la red a estar más saturada y por ende, los mensajes se demoran más en llegar a los nodos.

5.4.3. EFECTO DEL FLUJO DE EVENTOS SOBRE LA TOLERANCIA A FALLAS

En los resultados que se presentan en la Tabla 5.6, se puede ver el efecto que tiene el flujo de eventos sobre la tolerancia a fallas, donde a mayor es el flujo, mayor será la cantidad de eventos perdidos. Esto se debe a que los nodos se encuentran más sobrecargados y por tanto, tener sus colas de procesamiento más congestionadas, por lo que si un nodo de estos falla se pierda una mayor cantidad de eventos.

Por otro lado, cabe mencionar que el aumento que tiene la cantidad de mensajes de OVERLOAD para las simulaciones con 50 eps, respecto a los valores obtenidos para los otros dos flujos, se debe a que los nodos se ven subcargados y por tanto, intentan de solicitar una mayor cantidad de flujo pero este ya es el máximo permitido para ellos, respecto a la cantidad de conexiones que posee el nodo *upstream* al que están conectados. Así mismo, se ve que baja la cantidad de nodos efectivamente fallidos debido a que se le ejerce más carga a los nodos y por tanto su batería se ve consumida de forma más rápida, lo que los llevaría a ser desactivados por el sistema y por ende, a medida que pasa el tiempo hay menos dispositivos conectados.

Tabla 5.6: Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del flujo de eventos sobre el balance de carga.

Fuente: Elaboración Propia						
	50 eps		100 eps		200 eps	
	Promedio	σ	Promedio	σ	Promedio	σ
Tiempo de recuperación [s]	7.36	3.51	7.25	3.58	8.33	3.20
Número de nodos fallidos	2.20	1.14	2.00	0.82	1.50	1.08
CHECKPOINTING	4206.70	1585.06	4122.10	1630.68	4338.50	1543.41
OVERLOAD	7501.60	1563.75	6343.20	827.74	7191.30	1152.23
DELETE	1916.20	583.13	1762.70	473.73	1567.40	545.91
UPDATE	11538.30	1618.67	11514.90	959.87	10517.60	1321.31
Eventos perdidos	192.30	92.95	304.00	135.57	1941.56	1365.07

5.4.4. EFECTO DEL TIPO DE GRAFO DE PROCESAMIENTO SOBRE LA TOLERANCIA A FALLAS

Los resultados que se presentan en la Tabla 5.7, presentan el comportamiento del sistema de tolerancia a fallas frente a distintos grafos de procesamiento. Donde se ve que existe un aumento significativo en la cantidad de eventos perdidos debido al tipo de topología. Principalmente, en la topología trifurcada utilizada en estas simulaciones, se ve que la cantidad de eventos perdidos se dispara con un aumento de un 763,34 % respecto a la cantidad de eventos perdidos en la topología lineal y 256 % con respecto a los perdidos en la topología bifurcada. Esto se debe a la configuración de los operadores en esta topología trifurcada, produciendo que los nodos siempre repliquen a los operados que se encuentran en las ramas de esta topología, ya que no tienen permitido replicar el operador que trifurca el flujo. Por esto, siempre están conectados a un nodo estático, lo que hace que el flujo que llega a ellos es mayor al que si estuvieran conectados a un nodo móvil.

Tabla 5.7: Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del grafo de procesamiento de sobrecarga sobre el balance de carga.

Fuente: Elaboración Propia						
	Lineal		Bifurcada		Trifurcada	
	Promedio	σ	Promedio	σ	Promedio	σ
Tiempo de recuperación [s]	7.25	3.58	7.46	3.12	7.59	3.23
Número de nodos fallidos	2.00	0.82	2.40	1.17	2.20	1.32
CHECKPOINTING	4122.10	1630.68	4114.80	1362.14	5159.00	2653.81
OVERLOAD	6343.20	827.74	6797.00	947.15	5741.00	1693.06
DELETE	1762.70	473.73	1750.20	483.50	1402.60	598.42
UPDATE	11514.90	959.87	11619.90	629.71	11241.80	553.36
Eventos perdidos	304.00	135.57	903.20	712.88	2320.56	2357.56

5.4.5. ANÁLISIS FINAL PARA LA TOLERANCIA A FALLAS

Como se puede observar en las mediciones realizadas de la cantidad de eventos perdidos y el tiempo de recuperación del sistema, se puede decir que el sistema responde de manera apropiada frente a estas fallas. En promedio se tiene que solo el 0.24 % de los eventos se pierde en el marco de estas simulaciones, obviamente esto dependerá del ambiente en que se ejecute el sistema, pero las variables utilizadas se tomaron para estar lo más apegadas posible a la realidad. Junto con esto, el sistema sabe cuales son los eventos que no han llegado por lo que es posible realizar el reenvío de estos para así asegurar una semántica de entrega de *at-least-once*.

Por otro lado, se tiene que, en promedio, el tiempo de recuperación del sistema en caso de falla, es de alrededor de los 7.5 segundos. Teniendo en cuenta que el RM cuenta con la información de todas las conexiones del sistema, este no cortará el flujo hacia los otros caminos que existen dentro de la topología, enfocándose solo en los nodos afectados por esta falla. Lo que ayuda a no cortar el flujo de eventos que es procesado por el sistema de manera global, si no de manera local, mejorando el rendimiento de este.

CONCLUSIONES

En este capítulo se presentan las conclusiones que se obtienen a partir del trabajo realizado en esta memoria. Posteriormente, se presenta las conclusiones generales y por último se exponen los trabajos futuros a ser realizados para la continuación de esta investigación.

Dentro del desarrollo de este trabajo se investigó las características de los sistemas de procesamiento de *stream*, cuales son sus fortalezas y sus debilidades. Se estudió las características que identifican a estos sistemas, como también los distintos sistemas que los componen. Principalmente, se estudió los sistemas de balance de carga y de tolerancia a fallas. Por parte de los sistemas de balance de carga, se estudiaron los distintos tipos de algoritmos para realizar esta tarea. Finalmente, se investigaron los sistemas de tolerancia a fallas, cuales son los algoritmos de recuperación de fallas, como también cuales son las semánticas de entrega de mensajes que aseguran cada uno de estos algoritmos y que es lo que conlleva cada una de estas.

Se vio que existe una gran variedad de tipos de sistemas de procesamiento de *stream*, los cuales pueden solucionar distintos problemas que necesiten procesar gran cantidad de datos en tiempo real. Estos utilizan distintos mecanismos para asegurar el procesamiento de la información lo que hace que cada uno funcione mejor que otro frente a alguna situación o escenario en específico.

Se abordaron las posibilidades de utilizar los dispositivos móviles para apoyar el poder de cómputo de sistemas ya existentes, aprovechando los principios que se presentan en el paradigma de *Fog Computing* y *Mobile Computing*. Permitiendo apoyar a un sistema de procesamiento de *stream* con dispositivos móviles los cuales asumen parte de la carga de estos sistemas, permitiendo que estos no se sobrecarguen, lo que mantiene la estabilidad de sus partes.

Se estudió un modelo de procesamiento de *stream* asistido con dispositivos móviles que permite agregar estos dispositivos de forma fluida y sin contratiempos al sistema para que aporten a los trabajos de procesamiento que se requieren. Administrando estos dispositivos para minimizar las fallas y maximizar el rendimiento de estos.

Se analizaron distintos tipos de simuladores con la necesidad de saber cuál era el que mejor se apegaba a las necesidades de este trabajo. En donde NS-3 presenta una flexibilidad y una fácil interfaz para poder realizar y desarrollar los módulos y simulaciones necesarias para lograr los objetivos planteados en esta memoria.

De los resultados obtenidos a través de la evaluación del modelo por medio de las simulaciones en NS-3 se concluye que las distintas partes del sistema cumplen con sus tareas, permitiéndole al sistema funcionar de forma adecuada. Específicamente, el sistema es capaz de balancear la carga entre los nodos estáticos y los nodos móviles, descongestionando las colas de procesamiento de los nodos estáticos a cambio de un pequeño aumento en la

latencia de procesamiento.

Se demostró que al ingresar nodos móviles al sistema este redistribuye la carga entre ellos, lo que no quiere decir que baje la carga del sistema ya que ella depende solamente del flujo de eventos que produce un operador *source*. Por parte de las topologías de procesamiento se observó que estas influyen sobre el modelo debido a que estas dictarán como se configuran las conexiones entre los nodos, lo que puede llevar a que algunos nodos se encuentren más sobrecargados que otro, por tanto esto se debe tener en cuenta al momento de desarrollar el sistema.

Se analizó y concluyó que para el caso del *overhead* del sistema, los tiempos de ciclo son los principales factores que pueden llegar a afectar a esto. Por lo que hay que poner en la balanza cual de estos dos se valoriza por sobre el otro. Donde a menor tiempo de ciclo mayor es el *overhead*, por tanto no se puede hablar de que el desempeño del sistema mejore a un menor tiempo de ciclo, por lo que se podría aseverar que es mejor que el sistema tenga un tiempo de ciclo más grande.

El sistema también es tolerante a fallas, debido a que las medidas de prevención todas por él son efectivas y disminuyen la cantidad de fallos que podría presentar el sistema, donde en promedio se tiene que solo se pierden un 0,24 % de los eventos procesados. En caso de falla el sistema es capaz de recuperarse de forma rápida y sin realizar grandes cortes al funcionamiento del sistema, además de llevar un registro de los eventos que llegan y tener la posibilidad de reenviar aquellos que no han sido procesados debido a un fallo.

OBJETIVOS GENERALES

Basando en lo estudiado se puede identificar falencias y/o oportunidades de mejoras para aumentar el desempeño del sistema, teniendo en cuenta las distintas características que posee el sistema, las cuales son el balance de carga, la tolerancia a fallas, la latencia de procesamiento, la pérdida de información, los tiempos de recuperación y el *overhead* del sistema. Las falencias encontradas fueron el modelo de transmisión de eventos, ignorar el efecto de bifurcaciones en la topología de procesamiento, el mecanismo de ingreso de nodos al sistema y el tipo de conexiones entre los *Operator Managers*. Por otro lado, las oportunidades de mejora encontradas fueron la utilización de un mapa virtual para la administración de los nodos que tenga en cuenta los recursos tanto los recursos móviles como los estáticos, el balance de carga de forma heterogénea y la implementación del umbral inferior de la carga de los nodos.

Se desarrolló pruebas para evaluar el desempeño de dicho modelo junto con las mejoras presentadas, teniendo en cuenta siempre el balance de carga, la tolerancia a fallas, la latencia de procesamiento, la pérdida de información, los tiempos de recuperación y el *overhead* del sistema. Estas evaluaciones consideraron individualmente cada parte del sistema para poder ver su funcionamiento de forma más precisa. Se evaluó el comportamiento de cada

parte frente a distintos escenarios, y también se evaluó cada una frente a distintas variables internas del sistema.

Se desarrolló un módulo en NS-3, que permite simular el modelo y controlar las variables que afectan al funcionamiento de él, se creó que la forma en que se pegara de la mejor forma a lo que sería una implementación real, emulando los distintos componentes del modelo y la relación entre ellos. Permitiendo así simular un sistema de procesamiento de *stream* y los múltiples dispositivos móviles que desean ingresar a él para procesar un flujo de eventos. Este módulo fue diseñado pensando en su reusabilidad para próximos proyectos, por lo que era necesario que fuera modular y tuviera en cuenta cada componente que forma el sistema, desde *Nimbus* hasta los *workers* en conjunto con los *operator managers*.

Se analizaron los resultados obtenidos por las evaluaciones y se pudo determinar el buen funcionamiento del sistema frente a los distintos escenarios propuestos. Logrando identificar los efectos y las respuestas entre las variables que afectan al sistema y como este se adapta a ellas.

TRABAJO FUTURO

Como trabajo futuro se sugiere realizar un estudio comparativo y detallado, en el cual se evalúe y permita identificar cuáles son los valores, o rangos de valores, de las variables de decisión de diseño que generan mejor desempeño en el modelo. Como ya se habló en estas conclusiones, variables como el tiempo de los ciclos no necesariamente ayudan al desempeño del sistema de gran manera al tener un tiempo de ciclo menor, lo que es algo contraintuitivo. Si no que al tener un mayor tiempo realizan su tarea de igual forma y disminuyen el *overhead* del sistema. Además, este estudio requiere que se revise el desempeño del sistema en una mayor gama de escenarios, que cuenten con distintas topologías que realmente se usen para solucionar problemas reales.

Junto con esto, se sugiere volver a realizar el *benchmarking* de los dispositivos móviles teniendo en cuenta una mayor cantidad de ellos, los cuales cuenten con una mayor variedad en sus características para evaluar el comportamiento del sistema frente a esta heterogeneidad y evaluar si es que es necesario mejorar el algoritmo de asignación de operadores para que el sistema mejore su rendimiento y se le saque mayor provecho a los recursos de los dispositivos móviles.

Finalmente, buscar que esta investigación termine en el desarrollo de este modelo en un ambiente real, el cual trabaje de forma unísona con *Storm* y se pueda implementar como una solución frente a algún problema en específico y que tenga ventajas competitivas frente a otros tipos de sistemas de procesamiento de *stream*.

REFERENCIAS BIBLIOGRÁFICAS

- [Agha, 1985] Agha, G. A. (1985). Actors: A model of concurrent computation in distributed systems. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB.
- [Akka, 2019] Akka (2019). Message delivery reliability. <https://doc.akka.io/docs/akka/current/general/message-delivery-reliability.html>.
- [Apache, 2018] Apache (2018). Companies using apache storm. <http://storm.apache.org/Powered-By.html>. Visitada: 19-07-2018.
- [Arasu et al., 2003] Arasu, Arvind and Babcock, Brian and Babu, Shivnath and Datar, Mayur and Ito, Keith and Motwani, Rajeev and Nishizawa, Itaru and Srivastava, Utkarsh and Thomas, Dilys and Varma, Rohit and others (2003). Stream: The stanford stream data manager. *IEEE Data Eng. Bull.*, 26(1):19–26.
- [Balazinska et al., 2005] Balazinska, M., Balakrishnan, H., Madden, S., y Stonebraker, M. (2005). Fault-tolerance in the borealis distributed stream processing system. En *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 13–24. ACM.
- [Banks, 1998] Banks, J. (1998). *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons.
- [Bonomi et al., 2014] Bonomi, F., Milito, R., Natarajan, P., y Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. En *Big data and internet of things: A roadmap for smart environments*, pp. 169–186. Springer.
- [Busching et al., 2012] Busching, F., Schildt, S., y Wolf, L. (2012). DroidCluster: Towards Smartphone Cluster Computing – The Streets are Paved with Potential Computer Clusters. En *32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 114–117.
- [Cardellini et al., 2015] Cardellini, V., Grassi, V., Presti, F. L., y Nardelli, M. (2015). On qos-aware scheduling of data stream applications over fog computing infrastructures. En *Computers and Communication (ISCC), 2015 IEEE Symposium on*, pp. 271–276. IEEE.
- [Casavant y Kuhl, 1988] Casavant, T. L. y Kuhl, J. G. (1988). A Taxonomy of Scheduling in General-purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, 14(2):141–154.
- [Cherniack et al., 2003] Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y., y Zdonik, S. B. (2003). Scalable distributed stream processing. En *CIDR*, volumen 3, pp. 257–268.

- [Comito *et al.*, 2017] Comito, C., Falcone, D., Talia, D., y Trunfio, P. (2017). Energy-aware task allocation for small devices in wireless networks. *Concurrency and Computation: Practice and Experience*, 29(1):e3831.
- [Dinh *et al.*, 2013] Dinh, H. T., Lee, C., Niyato, D., y Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611.
- [Foster *et al.*, 2008] Foster, I., Zhao, Y., Raicu, I., y Lu, S. (2008). Cloud computing and grid computing 360-degree compared. En *Grid Computing Environments Workshop*, 2008. GCE'08, pp. 1–10. Ieee.
- [Foundation, 2014] Foundation, A. S. (2014). S4 project incubation status. <http://incubator.apache.org/projects/s4.html>. Visitada: 02-09-2018.
- [Friedman y Tzoumas, 2016] Friedman, E. y Tzoumas, K. (2016). *Introduction to Apache Flink: Stream Processing for Real Time and Beyond*. O'Reilly Media, Inc.
- [Henderson, 2006] Henderson, T. (2006). Announcing a new project for ns-3. <https://mailman.isi.edu/pipermail/ns-developers/2006-July/002316.html>. Visitado 02-09-2018.
- [Henderson *et al.*, 2008] Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C., y Kopena, J. (2008). Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527.
- [Hernández Sampieri *et al.*, 2006] Hernández Sampieri, Roberto and Fernández Collado, Carlos and Baptista Lucio, Pilar and others (2006). *Metodología de la investigación*, volumen 3. México: McGraw-Hill.
- [Houlding, 2012] Houlding, B. (2012). General principles of discrete-event system simulation. <https://www.scss.tcd.ie/Brett.Houlding/Domain.sites2/sslides6.pdf>. Visitado 01-09-2018.
- [Hwang *et al.*, 2005] Hwang, J.-H., Balazinska, M., Rasin, A., Cetintemel, U., Stonebraker, M., y Zdonik, S. (2005). High-availability algorithms for distributed stream processing. En *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pp. 779–790, Washington, DC, USA. IEEE Computer Society.
- [I. Saidu *et al.*, 2014] I. Saidu, I., Subramaniam, S., y Jaafar, A. (2014). A load-aware weighted round-robin algorithm for ieee 802.16 networks. *EURASIP Journal on Wireless Communications and Networking*.
- [Intelligence, 2018] Intelligence, G. (2018). The mobile economy 2018. Technical report, GSMA.
- [Katevenis *et al.*, 1991] Katevenis, M., Sidiropoulos, S., y Courcoubetis, C. (1991). Weighted round-robin cell multiplexing in a general-purpose atm switch chip. *IEEE Journal on Selected Areas in Communications*, 9(8):1265–1279.

- [Kephart y Chess, 2003] Kephart, J. O. y Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.
- [MapR, 2015] MapR (2015). Apache spark. <https://mapr.com/products/product-overview/apache-spark/>. Visitada 24-07-2019.
- [Martin et al., 2015] Martin, A., Smaneto, T., Dietze, T., Brito, A., y Fetzer, C. (2015). User-Constraint and Self-Adaptive Fault Tolerance for Event Stream Processing Systems. En *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 462–473.
- [McGeoch, 1996] McGeoch, C. C. (1996). Toward an experimental method for algorithm simulation. *INFORMS Journal on Computing*, 8(1):1–15.
- [Morales, 2016] Morales, J. (2016). Modelo para la integración de recursos de mobile computing en sistemas distribuidos de procesamiento streaming. Tesis de máster, Universidad de Santiago de Chile.
- [Morales et al., 2014] Morales, J., Rosas, E., y Hidalgo, N. (2014). Symbiosis: Sharing mobile resources for stream processing. En *2014 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6. IEEE.
- [Nasir et al., 2015] Nasir, M. A. U., Morales, G. D. F., Garcia-Soriano, D., Kourtellis, N., y Serafini, M. (2015). The power of both choices: Practical load balancing for distributed stream processing engines. En *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pp. 137–148. IEEE.
- [Neumeyer et al., 2010a] Neumeyer, L., Robbins, B., Nair, A., y Kesari, A. (2010a). S4: Distributed stream computing platform. En *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pp. 170–177. IEEE.
- [Neumeyer et al., 2010b] Neumeyer, L., Robbins, B., Nair, A., y Kesari, A. (2010b). S4: Distributed Stream Computing Platform. En *IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 170–177, Sydney, NSW. Yahoo! Labs., Santa Clara, CA, USA.
- [Ning et al., 2015] Ning, Q., Chen, C.-A., Stoleru, R., y Chen, C. (2015). Mobile storm: Distributed real-time stream processing for mobile clouds. En *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pp. 139–145. IEEE.
- [Peng, 2016] Peng, B. J. (2016). Resource aware scheduling in apache storm. <https://www.slideshare.net/HadoopSummit/resource-aware-scheduling-in-apache-storm>. Visitada: 04-09-2018.
- [Pietzuch et al., 2006] Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., y Seltzer, M. (2006). Network-aware operator placement for stream-processing systems. En *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pp. 49–49. IEEE.

- [R. Kumar Singh y Saxena, 2012] R. Kumar Singh, N. S. C. y Saxena, K. (2012). "load balancing in ip/mps networks: A survey". *Communications and Network*, 4(2):151–156.
- [Riley y Henderson, 2010] Riley, G. F. y Henderson, T. R. (2010). The ns-3 network simulator. En *Modeling and tools for network simulation*, pp. 15–34. Springer.
- [Rychlý et al., 2015] Rychlý, M., Skoda, P., y Smrz, P. (2015). Heterogeneity-aware scheduler for stream processing frameworks. *IJBDI*, 2(2):70–80.
- [Shu et al., 2013] Shu, P., Liu, F., Jin, H., Chen, M., Wen, F., Qu, Y., y Li, B. (2013). etime: Energy-efficient transmission between cloud and mobile devices. En *INFOCOM, 2013 Proceedings IEEE*, pp. 195–199.
- [Siraj et al., 2012] Siraj, S., Gupta, A., y Badgujar, R. (2012). Network simulation tools survey. *International Journal of Advanced Research in Computer and Communication Engineering*, 1(4):199–206.
- [Spark, 2016] Spark, A. (2016). Spark Overview. <http://spark.apache.org/docs/latest/>. Visitada: 02-09-2018.
- [Storm, 2018] Storm (2018). Apache storm. <http://storm.apache.org>. Visitada: 04-08-2018.
- [Su y Zhou, 2015] Su, L. y Zhou, Y. (2015). Passive and Partially Active Fault Tolerance for Massively Parallel Stream Processing Engines. *CoRR*, abs/1508.04907.
- [Szpakowski, 2012] Szpakowski, M. (2012). Native port of the Boinc client for Android devices. <http://nativeboinc.org/site/uncat/start>. Visitada: 02-05-2016.
- [Toshniwal et al., 2014] Toshniwal, Ankit and Taneja, Siddarth and Shukla, Amit and Ramasamy, Karthik and Patel, Jignesh M and Kulkarni, Sanjeev and Jackson, Jason and Gade, Krishna and Fu, Maosong and Donham, Jake and others (2014). Storm@ twitter. En *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 147–156. ACM.
- [Wehrle et al., 2010] Wehrle, K., Günes, M., y Gross, J. (2010). *Modeling and tools for network simulation*. Springer Science & Business Media.
- [Zaharia et al., 2010] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., y Stoica, I. (2010). Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. En *Proceedings of the 5th European Conference on Computer Systems, EuroSys '10*, pp. 265–278, New York, NY, USA. ACM.

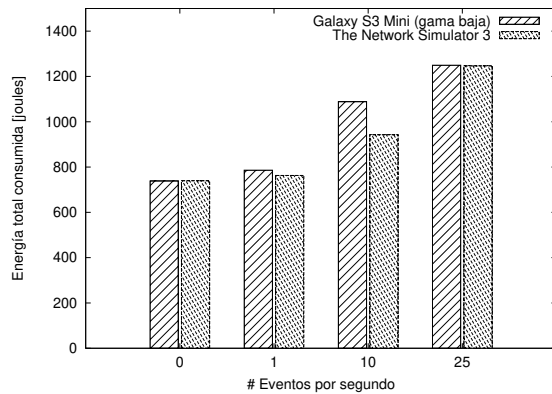
ANEXO

ANEXO: PARÁMETROS DE SIMULACIONES

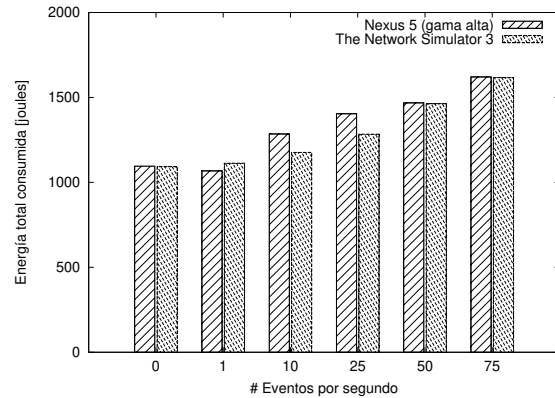
Tabla A.1: Latencia de procesamiento utilizada para simular poder de cómputo de los nodo en NS-3

Nodo	Aplicación	Latencia de procesamiento (ms)
Estático	Texto	2 - 6
Estático	Imagen	4 - 8
Móvil gama alta	Texto	8 - 18
Móvil gama alta	Imagen	15 - 25
Móvil gama baja	Texto	35 - 45
Móvil gama baja	Imagen	35 - 45

Fuente: [Morales, 2016].



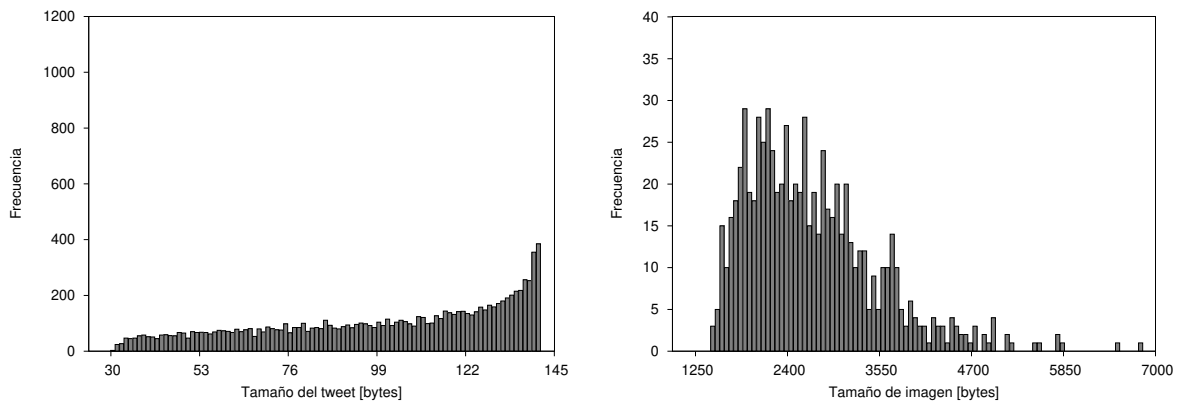
(a) Ajuste de energía en dispositivo móvil de gama baja



(b) Ajuste de energía en dispositivo móvil de gama alta

Figura A.1: Ajuste para simulación del consumo de energía en dispositivo móvil en NS-3.

Fuente: [Morales, 2016].



(a) Distribución Beta = $31 + 110\beta[1.18, 0.667]$, utilizada en NS-3 para el tamaño de los tweets en las simulaciones

(b) Distribución Gamma = $1430 + \Gamma[2.26, 552]$, utilizada en NS-3 para el tamaño de las imágenes en las simulaciones

Figura A.2: Ajuste para simulación del tamaño de eventos generados en NS-3.

Fuente: [Morales, 2016].

ANEXO: COMPLEMENTO DE RESULTADOS

A continuación, se muestran tablas y figuras complementarias a los resultados mostrados en Capítulo 5.

Tabla B.1: Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del flujo de eventos sobre la utilización de nodos.

Fuente: Elaboración Propia

<i>CHECKPOINTING</i>	1058
<i>OVERLOAD</i>	1066
<i>DELETE</i>	0
<i>UPDATE</i>	1064

Tabla B.2: Tabla de mensajes adicionales enviados durante las pruebas para evaluar el efecto del grafo de procesamiento sobre el sistema de balance de carga.

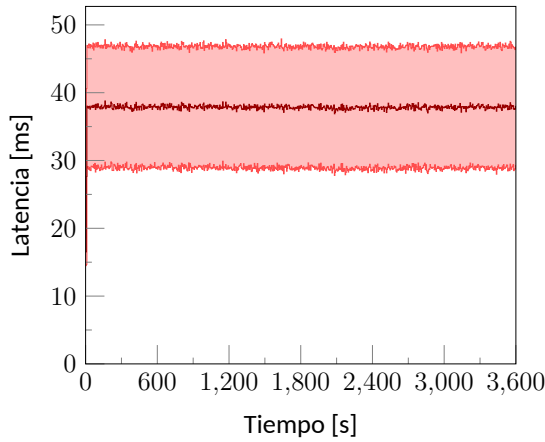
Fuente: Elaboración Propia

	Lineal	Bifurcada	Trifurcada
<i>CHECKPOINTING</i>	1058	1058	794
<i>OVERLOAD</i>	1083	714	482
<i>DELETE</i>	0	0	0
<i>UPDATE</i>	1064	1064	1064

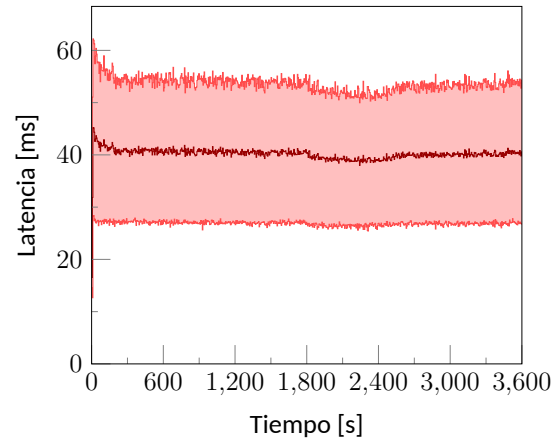
EFFECTO DEL UMBRAL DE ENERGÍA SOBRE LA TOLERANCIA A FALLAS

Los resultados que se presentan en la Tabla B.3 no muestran alguna relación entre los datos obtenidos y el cambio en la variable del umbral de energía. Bajo la lógica, se podría esperar que a mayor sea el umbral menor sea la cantidad de eventos perdidos y en caso contrario, que a menor sea el umbral mayor sea la cantidad de eventos perdidos. Debido que a menor sea el umbral menor es el espacio que se le da al nodo entre que este activo y que su batería se vea agotada, lo que causaría pérdida de eventos. Pero este efecto no se ve reflejado en los datos. Por otro lado, se esperaría que haya también una cantidad mayor de número de mensajes *DELETE*, pero esto tampoco se aprecia.

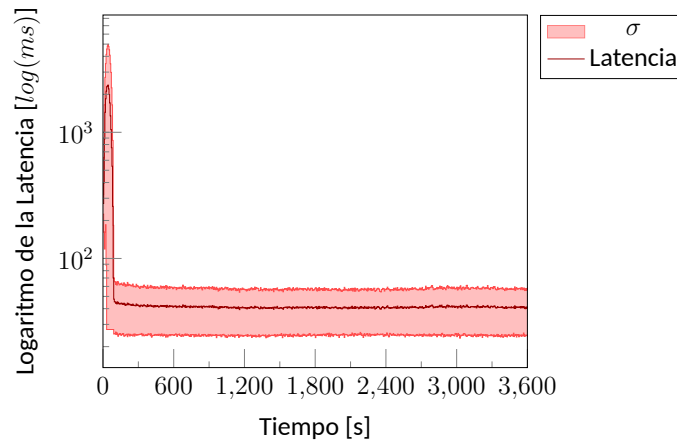
Solo se puede ver un pequeño aumento en la cantidad de nodos efectivamente fallidos dentro de las simulaciones, lo que se explicaría por que al disminuir el umbral existen más nodos activos dentro del sistema, y, por tanto, es más probable que al fallar un nodo, este sea un nodo activo. Esto último, no se ve que tuviera algún efecto en la cantidad de eventos perdidos dentro de las simulaciones.



(a) Flujo de 50 eventos por segundo.



(b) Flujo de 100 eventos por segundo.



(c) Flujo de 200 eventos por segundo.

Figura B.1: Gráficos de Latencia vs Tiempo de los resultados de los experimento para observar los efectos del flujo de eventos sobre el balance de carga.

Fuente: Elaboración Propia.

EFFECTO DE LA GAMA DEL DISPOSITIVO MÓVIL SOBRE LA TOLERANCIA A FALLAS

En la Tabla B.4 de los resultados obtenidos para ver el efecto de la gama del dispositivo sobre la tolerancia a fallas, se observa que hay tanto un aumento en el tiempo de recuperación, como también en la cantidad de nodos efectivamente fallidos con respecto al aumento en la gama. Esto es debido a que un dispositivo de mayor gama, es decir, con mejores recursos que uno de gama baja, tiende a estar mayor cantidad de tiempo dentro del sistema, ya sea por señal o batería. Lo que también se ve reflejado tanto en la cantidad de mensajes de *UPDATE* que se envían en promedio en las simulaciones, como también la disminución del envío de mensajes de *OVERLOAD*, que son los mensajes de sobrecarga. Esto es debido a que los nodos de mayor gama se ven menos sobrecargados por el flujo de eventos.

Por otro lado, se puede ver que no existe una correlación evidente entre la gama y la canti-

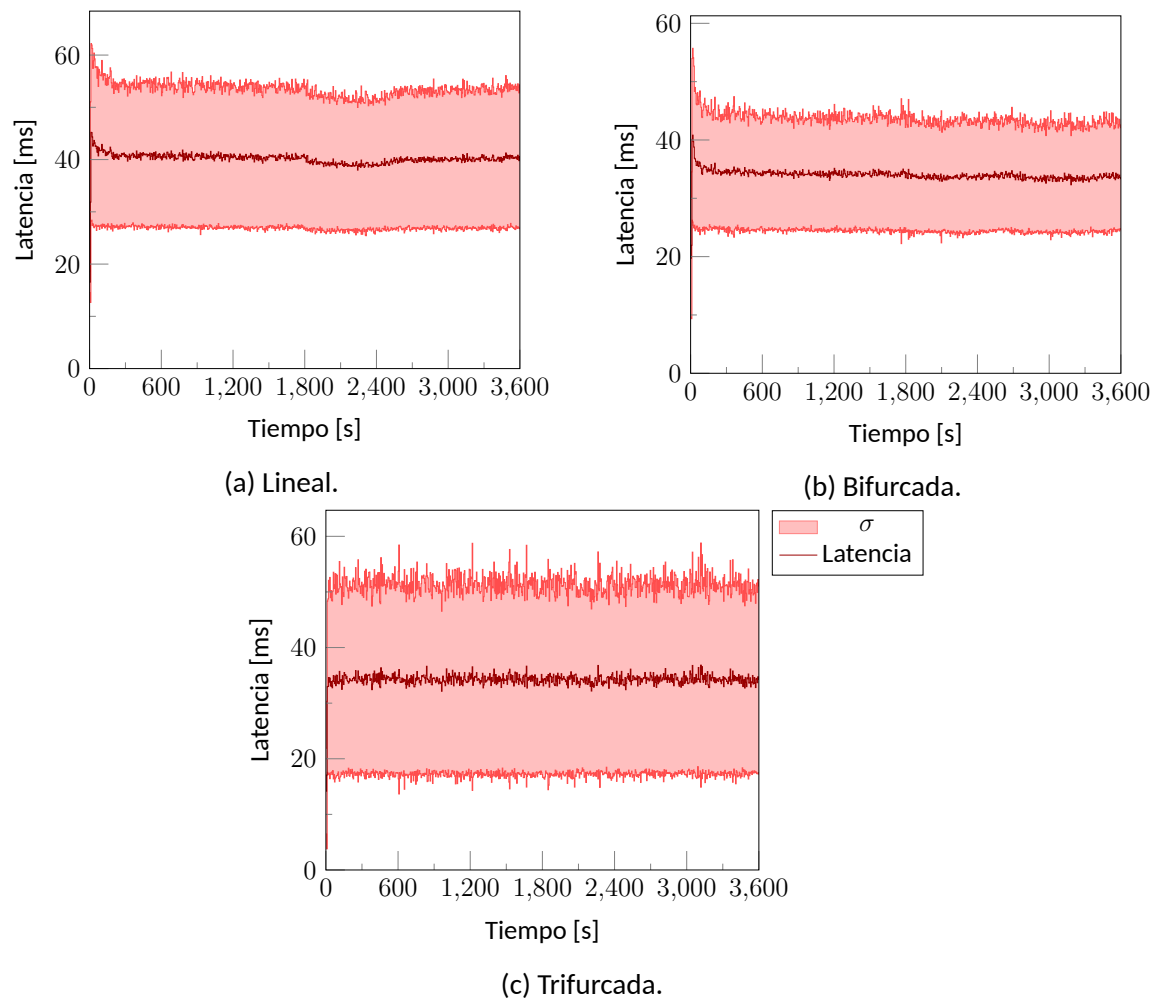


Figura B.2: Gráficos de Latencia vs Tiempo de los resultados de los experimento para observar los efectos del tipo de grafo de procesamiento sobre el balance de carga.

Fuente: Elaboración Propia.

dad de eventos perdidos o el tiempo de recuperación. A pesar de que para las simulaciones realizadas con dispositivos mixtos, es decir, se realizaron con dispositivos tanto de gama alta como de gama baja, se ve una baja en los valores medidos tanto de tiempo de recuperación como de perdida de eventos. Por lo que cabe recordar que este valor tuvo que ser trabajado, eliminando un *outlier*, lo que influencio esta medición.

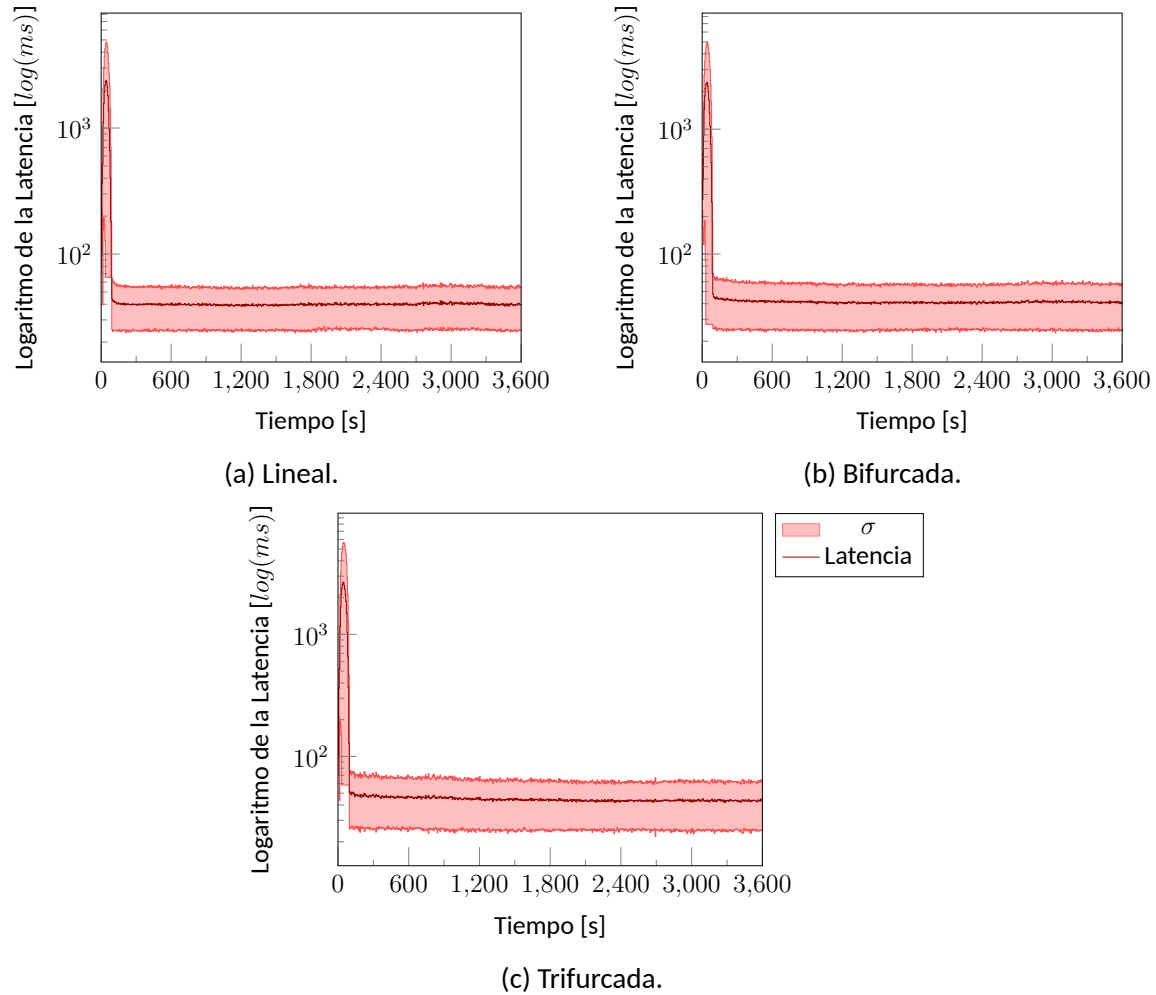


Figura B.3: Gráficos de Latencia vs Tiempo de los resultados de los experimento para observar los efectos del tipo de grafo de procesamiento sobre el balance de carga.

Fuente: Elaboración Propia.

Tabla B.3: Tabla resumen de los datos medidos en las pruebas para evaluar el efecto del umbral de energía sobre la tolerancia a fallas.

Fuente: Elaboración Propia

	Energía > 0.05		Energía > 0.15		Energía > 0.25	
	Promedio	σ	Promedio	σ	Promedio	σ
Tiempo de recuperación [s]	7.43	3.59	7.25	3.58	7.25	3.58
Número de nodos fallidos	2.10	0.99	2.00	0.82	2.00	0.82
CHECKPOINTING	3733.30	1319.80	4122.10	1630.68	4071.80	1610.96
OVERLOAD	5866.60	675.49	6343.20	827.74	6311.80	815.00
DELETE	1762.40	537.57	1762.70	473.73	1795.20	536.60
UPDATE	10251.10	1108.90	11514.90	959.87	11381.20	1452.90
Eventos perdidos	319.70	166.16	304.00	135.57	386.00	238.07

Tabla B.4: Tabla resumen de los datos medidos en las pruebas para evaluar el efecto de la gama del dispositivo móvil sobre la tolerancia a fallas.

Fuente: Elaboración Propia

	Gama Baja		Mixta		Gama Alta	
	Promedio	σ	Promedio	σ	Promedio	σ
Tiempo de recuperación [s]	7.05	3.57	7.25	3.58	7.30	3.44
Número de nodos fallidos	1.90	1.10	2.00	0.82	2.30	1.16
CHECKPOINTING	3752.60	1436.23	4122.10	1630.68	4546.60	1640.93
OVERLOAD	7428.00	1634.44	6343.20	827.74	5631.40	1148.31
DELETE	1611.10	554.06	1762.70	473.73	1942.60	575.25
UPDATE	10986.90	1195.62	11514.90	959.87	11856.40	1125.96
Eventos perdidos	437.11	245.58	304.00	135.57	458.10	263.95