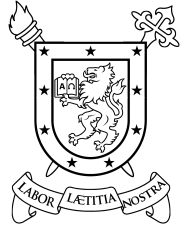


UNIVERSIDAD DE SANTIAGO DE CHILE  
FACULTAD DE INGENIERÍA  
Departamento de Ingeniería Informática



Algoritmos de predicción y distribución de carga para el grafo lógico en  
los sistemas de procesamiento de *stream*

Daniel Pedro Pablo Wladdimiro Cottet

Profesor guía: Nicolás Hidalgo Castillo

Profesor co-guía: Erika Rosas Olivos

Tesis de grado presentada en  
conformidad a los requisitos para  
obtener el grado de Magíster en  
Ingeniería Informática

Santiago – Chile

2015



© Daniel Pedro Pablo Wladdimiro Cottet - 2015



• Algunos derechos reservados. Esta obra está bajo una Licencia Creative Commons Atribución-Chile 3.0. Sus condiciones de uso pueden ser revisadas en:  
<http://creativecommons.org/licenses/by/3.0/cl/>.

*Dedicado...*

## AGRADECIMIENTOS

Agradezco a



# TABLA DE CONTENIDO

|   |             |
|---|-------------|
| <b>Índice de Figuras</b>                            | <b>ix</b>   |
| <b>Índice de Algoritmos</b>                         | <b>xi</b>   |
| <b>Resumen</b>                                      | <b>xiii</b> |
| <b>Abstract</b>                                     | <b>xv</b>   |
| <b>1 Introducción</b>                               | <b>1</b>    |
| 1.1 Antecedentes y motivación . . . . .             | 1           |
| 1.2 Descripción del problema . . . . .              | 3           |
| 1.3 Solución propuesta . . . . .                    | 3           |
| 1.4 Objetivos y alcance del proyecto . . . . .      | 5           |
| 1.4.1 Objetivo general . . . . .                    | 5           |
| 1.4.2 Objetivos específicos . . . . .               | 5           |
| 1.4.3 Alcances . . . . .                            | 5           |
| 1.5 Metodología y herramientas utilizadas . . . . . | 6           |
| 1.5.1 Metodología . . . . .                         | 6           |
| 1.5.2 Herramientas de desarrollo . . . . .          | 7           |
| 1.6 Resultados Obtenidos . . . . .                  | 7           |
| 1.7 Organización del documento . . . . .            | 7           |
| <b>2 Marco Teórico</b>                              | <b>9</b>    |
| 2.1 Streaming . . . . .                             | 9           |
| 2.2 Stream processing . . . . .                     | 10          |
| 2.3 Sistemas de Procesamiento de Stream . . . . .   | 11          |
| 2.4 Elasticidad . . . . .                           | 15          |
| 2.5 Procesos estocásticos . . . . .                 | 16          |
| 2.5.1 Cadena de Markov . . . . .                    | 17          |
| 2.5.2 Trabajo relacionado . . . . .                 | 19          |
| 2.6 Teoría de colas . . . . .                       | 20          |

|   |           |
|---|-----------|
| <b>3 Balance de carga en SPS</b>                            | <b>23</b> |
| 3.1 Perspectivas de balance de carga . . . . .              | 23        |
| 3.1.1 Recursos físicos . . . . .                            | 23        |
| 3.1.2 Recursos lógicos . . . . .                            | 24        |
| 3.1.3 Enfoque estático . . . . .                            | 24        |
| 3.1.4 Enfoque dinámico . . . . .                            | 26        |
| 3.2 Técnicas de balance de carga . . . . .                  | 27        |
| 3.2.1 Planificación determinista . . . . .                  | 27        |
| 3.2.2 Load Shedding . . . . .                               | 28        |
| 3.2.3 Migración . . . . .                                   | 29        |
| 3.2.4 Fisión . . . . .                                      | 30        |
| <b>4 Diseño del sistema de distribución de carga</b>        | <b>33</b> |
| 4.1 Análisis del sistema de distribución de carga . . . . . | 33        |
| 4.2 Recolección de los datos . . . . .                      | 37        |
| 4.3 Algoritmo reactivo . . . . .                            | 38        |
| 4.4 Algoritmo predictivo . . . . .                          | 40        |
| 4.5 Administración del sistema . . . . .                    | 43        |
| <b>5 Experimentos y evaluación</b>                          | <b>45</b> |
| 5.1 Implementación del sistema . . . . .                    | 45        |
| 5.2 Experimentos . . . . .                                  | 47        |
| 5.2.1 App 1 . . . . .                                       | 47        |
| 5.2.2 App 2 . . . . .                                       | 47        |
| 5.2.3 App 3 . . . . .                                       | 48        |
| 5.3 Evaluación . . . . .                                    | 48        |
| <b>Referencias</b>  | <b>49</b> |
| <b>Anexos</b>   | <b>53</b> |
| <b>A Conformación de matriz de transición</b>               | <b>55</b> |
| <b>B Modificaciones al código fuente de S4</b>              | <b>57</b> |



## ÍNDICE DE FIGURAS

|            |  |    |
|------------|--|----|
| Figura 2.1 | Flujo de datos entre el servidor y los clientes. . . . .                   | 9  |
| Figura 2.2 | Ejemplo de modelo de SPS. . . . .  | 12 |
| Figura 2.3 | Modelo push. . . . .   | 14 |
| Figura 2.4 | Modelo pull. . . . .   | 15 |
| Figura 2.5 | Elasticidad en un SPS. . . . .   | 16 |
| Figura 2.6 | Proceso de Markov. . . . .   | 17 |
| Figura 2.7 | Cadena de Markov. . . . .  | 18 |
| Figura 2.8 | Ejemplo de cadena de Markov. . . . .                                       | 18 |
| Figura 2.9 | Ejemplo de un sistema basado en teoría de colas. . . . .                   | 21 |
| Figura 3.1 | Load shedding en un SPS. . . . .   | 29 |
| Figura 3.2 | Técnica de migración en un SPS. . . . .                                    | 30 |
| Figura 3.3 | Técnica de fisión en un SPS. . . . .                                       | 31 |
| Figura 3.4 | Ejemplo de replicación de los operadores (Fernandez et al., 2013). . . . . | 32 |
| Figura 4.1 | Ejemplo de replicación del sistema propuesto. . . . .                      | 34 |
| Figura 4.2 | Enfoque de un SPS con conceptos de teoría de colas. . . . .                | 35 |
| Figura 4.3 | Estructura del sistema de distribución de carga. . . . .                   | 36 |
| Figura 4.4 | Comportamiento de la tasa de procesamiento de un operador. . . . .         | 39 |
| Figura 4.5 | Cadena de Markov dado el modelo propuesto del sistema. . . . .             | 41 |
| Figura 5.1 | Distribución de la carga entre las réplicas. . . . .                       | 46 |



## ÍNDICE DE ALGORITMOS

|     |  |    |
|-----|--|----|
| 4.1 | Algoritmo reactivo del sistema de distribución de carga. . . . .   | 39 |
| 4.2 | Cálculo de la distribución estacionaria de la cadena de Markov de un<br>operador $\phi$ . . . . .                          | 42 |
| 4.3 | Algoritmo predictivo del sistema de distribución de carga. . . . .   | 43 |
| 4.4 | Administración de réplicas de un operador $\phi$ dado su comportamiento<br>en el sistema de distribución de carga. . . . . | 44 |
| 5.1 | Distribución de carga entre las réplicas de un operador. . . . .   | 45 |
| A.1 | Algoritmo para la conformación de la matriz de transición. . . . .   | 55 |



# RESUMEN

resumenBlaBlaBla

**Palabras Claves:** SPS;Elasticidad;Algoritmos reactivos



# ABSTRACT

abstractBlaBla

**Keywords:** SPS;Elastic;Algorithm reactive;Algorithm predictive





# CAPÍTULO 1. INTRODUCCIÓN

## 1.1 ANTECEDENTES Y MOTIVACIÓN

La gran contribución de información en la Internet se ha debido al origen de la Web 2.0, donde ésta se caracteriza por la participación activa del usuario, siendo reflejado en el auge de blogs, redes sociales u otras aplicaciones web (Oberhelman, 2007). Debido a lo anterior, se crean sistemas de procesamiento para grandes cantidades de información generadas por la interacción entre los usuarios.

Es así como con el tiempo se han ido creando distintas aplicaciones de *streaming*, debido al interesante funcionamiento que poseen, las que se caracterizan por ser capaces de procesar grandes flujos de datos en tiempo real (Chen & Zhang, 2014). La necesidad de procesar información en tiempo real surge dado que muchas aplicaciones, donde sus usuarios requieren de respuestas rápidas y actualizadas que le permitan tomar decisiones en períodos cortos de tiempo. Dentro de los ejemplos existentes se encuentran; análisis de sentimientos de los mensajes de usuarios, análisis de los precios de la bolsa de valores, recopilación de información en caso de emergencia, entre otros. Las distintas aplicaciones que se han creado se volvieron críticas para sus usuarios, debido que sustenta la toma de decisiones de empresas o instituciones (Wenzel, 2014).

Un ejemplo de esto, es aplicaciones que analizan las redes sociales en caso de un desastre natural, donde grandes cantidades de información son procesadas, procesando esta información lo más cercano al tiempo real para obtener información que sea relevante para la situación (Andrade et al., 2014). De esta manera, se puede construir un sistema distribuido que pueda procesar los datos realizando análisis de sentimiento, búsqueda de palabras claves o filtros de búsqueda, ya sea por idioma, país o género, realizando un procesamiento lo más cercano al tiempo real. Con esta información, se puede realizar análisis de sectores críticos, búsqueda de personas o aviso de alertas, lo cual sería crucial para tomar decisiones en estos momentos.

Por otra parte, también es utilizado estos sistemas de procesamiento para predicciones en la bolsa de comercio, de esta manera, se crean sistemas de

procesamiento de los datos que vayan llegando en el día, de tal manera con estos datos existan modelos matemáticos que predican el comportamiento para el siguiente día. Con estos sistemas, la ganancia que existe por parte de las personas interesadas puede aumentar considerablemente, por lo que ha generando un alto interés en el desarrollo e investigación en esta área con el transcurso del tiempo.

También se aplica en casos de seguridad, dado que se realiza un monitoreo de la actividad que surge por parte de los usuarios que interactúan en una red específica. Esto es útil para empresas o ministerios que poseen información privilegiada, y en caso que alguien desee realizar respaldos o eliminar información sin consentimiento de los encargados, puede detectarse la persona y generarse una alarma de preventiva a las autoridades. Como la información es procesada en tiempo real, la cantidad de datos que pueden irse procesando hace que el retraso de la información procesada sea baja, de esta manera, ayuda a detectar a tiempo las posibles acciones de usuarios maliciosos.

Entre los sistemas actuales de procesamiento de *stream* se encuentran S4 (Neumeyer et al., 2010), Storm (Storm, 2014), Samza (Samza, 2014), entre otros, los cuales son los más utilizados como arquitectura de procesamiento en la confección de distintas aplicaciones de *streaming*. Aunque poseen bastante flexibilidad para la creación de un sistema, por la facilidad de crear distintas topologías, no lo tiene para adaptarse en el tiempo, debido a que las topologías de procesamiento generadas son estáticas, por lo que dada la naturaleza dinámica de las interacciones pueden surgir problemas de sobrecarga.

El problema de sobrecarga conlleva a una baja en el rendimiento, produciendo una pérdida de recursos, tiempo o información. Abortar este problema es crítico, puesto que implica una mejora en la exactitud y disminución en el tiempo de procesamiento, debido que al tener mayor cantidad de datos, menor tiempo de procesamiento, se mejora la información entregada. Un ejemplo de esto, es que se posee un tiempo  $t$  para procesar  $n$  datos, de disminuir el tiempo de procesamiento total de los datos, se tendrá que en el mismo tiempo  $t$  se procesarán una cantidad  $n + m$  de datos, donde  $m$  son los datos adicionales a analizar debido a la mejora del rendimiento. Como existe una mejora en la cantidad de datos para analizar, la

información de salida es más exacta, debido que tiene más datos con que comparar. De esta manera, se efectúa una mejora en los recursos utilizados, debido a la disminución del tiempo de procesamiento, y una mejor calidad en la información entregada al usuario.

## 1.2 DESCRIPCIÓN DEL PROBLEMA

Los SPS (Sistemas de Procesamiento de *Streaming*) están planteados como un grafo cuyas vértices son operadores y las aristas son flujo de datos entre los operadores. Dada su representación, puede existir sobrecarga del sistema a producto de factores físicos o lógicos. El factor físico se define como los componentes que posee la máquina, los cuales pueden ser limitantes para el sistema alojado. En cambio, el lógico se concentra en los componentes del grafo, por lo que existe una limitante en la cantidad de operadores o la cantidad de flujo existente entre los operadores.

Debido a lo anterior, existe un problema en el sistema a raíz de la sobrecarga que puede darse en el sistema debido a los distintos factores lógico, como la cola de cada operador, por la falta de flexibilidad del SPS en los operadores más demandados. Esto sucede dada la condición estática del grafo, es decir que no varía la topología del grafo con el tiempo, por lo que no existe una forma de disminuir la carga y reducir las colas, de tal manera de mejorar el rendimiento del sistema y obtener información cercana al tiempo real.

## 1.3 SOLUCIÓN PROPUESTA

La solución propuesta consiste en el diseño de algoritmos de predicción y distribución de carga a nivel de la lógica del grafo. Por lo que propone implementar cuatro módulos que componen la estructura del sistema de distribución de carga para el diseño de los algoritmos, los cuales se componen por el monitor de carga, analizador de carga, predictor de carga y administrador de réplicas.

El monitor de carga está encargado de recuperar el nivel de carga de cada uno de los operadores. Esta información es entregada a otros dos módulos, los cuales están encargados de procesarla de tal manera de ver si existe alguna sobrecarga. Cada uno de éstos trabaja de forma independiente y tiene distintos métodos, uno proactivo y otro reactivo, de tal manera de poseer mayor exactitud en la detección de una sobrecarga.

El analizador de carga consiste en un método reactivo, el cual analiza el tráfico de los operadores en el tiempo actual, y cuantifica su carga. La sobrecarga de cada operador depende de un umbral, por lo que según esto se envía al administrador de réplica el tráfico de cierto operador de ser necesario una replicación.

El predictor de carga consiste en un método proactivo, el cual analiza la carga de los distintos operadores según una ventana de tiempo, y predice la carga según un método predictivo. De esta manera, se determina la posible carga que existe en cierto período de tiempo futuro, donde según un umbral y un margen de error se envía el tráfico de carga de un operador al administrador de réplicas, y así analizar si es necesario una replicación.

El administrador de réplicas se alimenta de la información entregada por los dos módulos anteriores, y así toma una decisión de la administración de cada una de las réplicas de los distintos operadores. Por lo tanto, verifica cuántas réplicas son necesarias según la cantidad de tráfico de cierto operador.

Finalmente, el sistema de procesamiento constantemente está realizando un *feedback* al sistema de optimización, de tal manera que pueda administrar las réplicas necesarias. De esta manera, se poseerá un sistema procesa información de manera más rápida, a través de este sistema de optimización con bajo *overhead*.

## 1.4 OBJETIVOS Y ALCANCE DEL PROYECTO

### 1.4.1 Objetivo general

Diseño, construcción y evaluación de un algoritmo de predicción y un algoritmo de distribución de carga para sistemas de procesamiento de *stream*.

### 1.4.2 Objetivos específicos

1. Diseñar e implementar un algoritmo de predicción que permita estimar la carga de los operadores.
2. Diseñar e implementar un algoritmo de distribución que permita la administración de los operadores del grafo de procesamiento de forma elástica.
3. Diseñar y construir experimentos que permitan validar la hipótesis formulada.
4. Evaluar y analizar el rendimiento del sistema a través de aplicaciones generadas sobre sistemas de procesamiento de *stream*.

### 1.4.3 Alcances

Dentro de los alcances y limitaciones que se tienen en el proyecto son:

- La evaluación de la solución presentada se implementará sobre un solo sistema de procesamiento de *stream*.
- Se evalúe con al menos una aplicación bajo escenarios simulados utilizando datos reales.
- La distribución de flujo de datos es a nivel de operadores y no de nodos físicos, por lo que no se analizó la carga de estos últimos.

- Los algoritmos propuestos no incluyen técnicas que garanticen el procesamiento de todo el flujo de datos.
- En la evaluación de los algoritmos propuestos se consideró el costo de comunicación de manera igualitaria para todos los operadores.

## 1.5 METODOLOGÍA Y HERRAMIENTAS UTILIZADAS

### 1.5.1 Metodología

Dado el carácter de investigación de la propuesta de tesis, se propone utilizar el método científico para la realización de ésta. Dentro de las etapas propuesta por (Hernández Sampieri et al., 2010) están:

1. Formulación de la hipótesis: “La utilización de un modelo híbrido de paralelización que permitirá mejorar la distribución de carga entre los operadores de manera dinámica, logrando reducir los tiempos de procesamiento y pérdida de eventos”.
2. Elaboración del marco teórico: Exponer las investigaciones que existen sobre problemas de sobrecarga en los operadores de SPS. Así mismo, los conceptos fundamentales de estos sistemas.
3. Seleccionar el diseño apropiado de investigación: Diseñar el experimento para el problema de balance de carga a nivel lógico en un SPS, vale decir, los algoritmos de predicción y distribución. Cada ejecución de los experimentos se basan según los principios de un SPS.
4. Analizar los resultados: De deberá analizar los resultados según las estadísticas entregadas y el modelo propuesto.
5. Presentar los resultados: Elaborar el reporte de investigación y presentar los resultados en gráficos y tablas.
6. Concluir en base a los resultados de la investigación.

### 1.5.2 Herramientas de desarrollo

Para el procesamiento de *stream* se utilizó Apache S4 0.6.0, por lo que fue necesario para su configuración Java SE Development Kit 7. Dentro esto, el lenguaje de programación de cada una de las estructuras del sistema desarrollado fue en Java, por lo que se trabajó sobre el IDE Eclipse Standard 4.4.2, y para el prototipo del modelo matemático se utilizó MATLAB 2014a. De forma complementaria, se utilizó Texmaker 4.1 para la confección de los distintos informes requeridos y la documentación correspondiente al trabajo.

## 1.6 RESULTADOS OBTENIDOS

Pam pam !

## 1.7 ORGANIZACIÓN DEL DOCUMENTO

Pam pam pam !





## CAPÍTULO 2. MARCO TEÓRICO

### 2.1 STREAMING

Streaming es una técnica para la transferencia de datos de forma continua, de tal manera que sea temporal y secuencial, cuyo funcionamiento se basa en el envío de datos por parte de un ente externo a un sistema de procesamiento de información, donde en caso de estar ocupado el servicio, se dejan los datos en cola. Generalmente, esto es utilizado en la interacción con la Web, como redes sociales o reproducción *online* de contenido multimedia. En la Figura 2.1 se muestra un servidor que emana un flujo de datos que llega a distintos clientes, donde cada uno de ellos procesa la información entrante, y en caso de estar ocupado el procesamiento, se guarda en un *buffer* los datos para posteriormente ser procesados.

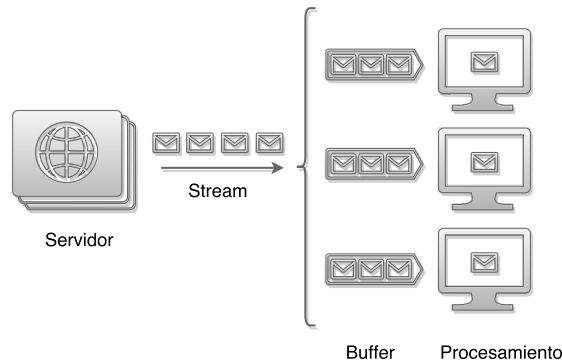


Figura 2.1: Flujo de datos entre el servidor y los clientes.

Este tipo de técnica es útil cuando se desea procesar información en tiempo real, siendo relevante la temporalidad de los datos, como la reproducción *online* de material multimedia. Los datos emanados por el *streaming* pueden ser utilizados para el análisis y procesamiento de una SPS (Sistema de Procesamiento de *Stream*). Un ejemplo de esto, es el *Streaming API* proporcionada por Twitter, donde esta información se puede utilizar para estudiar los *trending topic* o los *hashtag* más utilizados para casos específicos, como campañas electorales o desastres naturales.

## 2.2 STREAM PROCESSING

*Stream processing* es un paradigma de programación, en el cual está orientado al procesamiento de un flujo de datos en tiempo real. Se centra en la programación de aplicaciones que puedan procesar la información en el momento, utilizando los recursos del sistema de forma paralela o distribuida para cumplir su objetivo, de tal manera que su procesamiento sea lo más cercano al tiempo real.

Dentro de las aplicaciones existentes en el procesamiento de *stream*, están el monitoreo de signos vitales, detección de fraudes, reproducción de videos *online*, y para cada uno de ellos es necesario cumplir con ciertas características para el funcionamiento correcto del sistema. Para ello, se han propuesto ciertos requerimientos para el procesamiento continuo de datos (Andrade et al., 2014), los cuales serán desglosados a continuación:

- **Grandes cantidades de procesamiento de datos distribuidos** Esto significa que al tratar de procesar los datos, no se puede guardar en una base de datos y luego procesarlos, como en general lo realizan los *batch processing*, por lo tanto es necesario otro mecanismo que pueda procesarlos mientras va llegando la información entrante. Por lo que utilizar *stream processing* soluciona este problema, dado que la información entrante es procesada a medida que van llegando los datos.
- **Estrictas limitaciones de ancho de banda y latencia** Se refiere a la comunicación que existe por parte del proveedor de datos, de tal manera que no sea una limitante en el procesamiento de los datos el ancho de banda o la latencia que existe. Esto es importante, dado que no sirve un sistema de estimación de la bolsa del mercado que a producto de la latencia existente, envíe datos obsoletos. Siempre se debe mantener una baja latencia, para poseer los datos lo más cercano al tiempo real.
- **Procesamiento de datos heterogéneos** En su mayoría, los datos poseen distintos formatos, contenidos y niveles de ruido, por lo que es necesario realizar una normalización de estos, de tal manera de estandarizar el procesamiento.

- **Proporcionar alta disponibilidad a largo plazo** Es importante poseer un constante flujo de información, que sea estable y persistente en el tiempo, de tal manera que esté procesando constantemente los datos para el propósito designado. Por ejemplo, si se posee un sistema de análisis de partículas en el espacio, es necesario que posea una tolerancia a falla, para que en el caso que exista una anomalía, siga manteniéndose una disponibilidad por parte del sistema, y no se pierda el objetivo de observar en tiempo real y procesar esa información entrante.

## 2.3 SISTEMAS DE PROCESAMIENTO DE STREAM

Entre los diferentes motores de procesamiento de datos masivos, existen los sistemas de procesamiento de *stream*, los cuales reciben grandes cantidades de datos que deben procesar de forma distribuida y *online*. Para realizar esto, se requiere un cambio en el paradigma *batch processing*, el cual guarda los datos en una base de datos, los que posteriormente son procesados de forma *offline* (Hawwash & Nasraoui, 2014), a uno que procese de forma *online*. El paradigma utilizado se basa en grafos, donde los operadores corresponden a las vértices del grafo, y las aristas a los flujos de datos que salen del operador, siendo los datos proporcionados por un ente externo, ya sea *streaming* de datos de redes sociales, estadísticas del monitoreo de un sistema u otra información deseada en tiempo real (Shahrivari, 2014).

El modelo de procesamiento que se muestra en la Figura 2.2, corresponde a un SPS (Sistema de Procesamiento de *Stream*). Como se había mencionado anteriormente, los vértices corresponden a operadores, como por ejemplo analizadores de sentimientos, filtros de palabras o algún algoritmo en particular, y las aristas corresponden a los flujos de datos entre un operador y otro. Además de esto, se tiene una fuente de datos, la cual entrega los datos iniciales a los primeros operadores del grafo (Appel et al., 2012).

Cabe destacar que al ser distribuido los SPS, cada uno de los vértices del grafo serán alojados en un nodo disponible en el ambiente que se esté almacenado

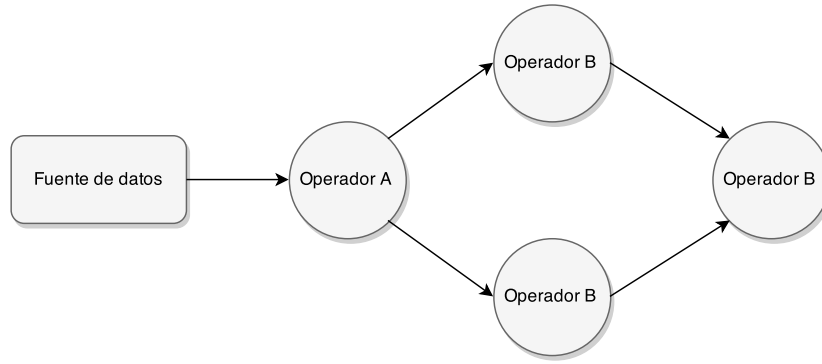


Figura 2.2: Ejemplo de modelo de SPS.

el sistema, ya sea un *cluster*, un *grid* o un *Infrastructure-as-Service*. Por lo tanto, se debe realizar una comunicación entre los distintos nodos, para realizar el envío del flujo de un operador a otro.

Los principales usos que se realizan en los SPS es el manejo de grandes cantidades de información, los cuales son procesados para obtener estadísticas o datos específicos de esto, como es el caso de detección de fraudes, recolección de información en caso de desastres o análisis de la interacción en las redes sociales. Para efectuar una procesamiento en tiempo real de los datos, se han propuesto los siguientes requerimientos (Stonebraker et al., 2005):

- **Baja latencia** Este concepto está asociado con la comunicación fluida entre los distintos nodos que estén trabajando en el sistema, de tal manera que no exista *delay* en el procesamiento del sistema.
- **Consultas SQL** Poder realizar consultas a una base de datos, sin perder las propiedades del SPS, como el procesamiento distribuido. Para esto, se debe realizar un cambio en la forma de ejecutar las consultas, debido que no sólo es necesario realizar la consulta, sino también realizar un *merge* de las respuestas, por lo que es necesario diseñar de una forma distinta el sistema a las formas convencionales.
- **Manejo de fallas en el flujo de dato** Esto significa que es importante poseer sistemas que no se preocupen de la pérdida en los datos, debido que se posee como premisa que se van a perder datos en el procesamiento de estos, ya sea por las colas, *delay* existente u otra anomalía existente. Por lo tanto, al esquematizar

el sistema es necesario lidiar con este tipo de fallas, ya sea diseñando márgenes de error u obviando esta pérdida en el procesamiento de los datos.

- **Generar resultados predecibles** Cuando se realizan consultas en el sistema, existe la posibilidad que sean correctas sólo por un período de tiempo, debido a alguna falla en el sistema que genere una pérdida en el estado del operador. Por lo tanto, es necesario garantizar que el resultado sea predecible y persistente en el tiempo, ya sea respaldando la información u otro mecanismo, de tal manera que si se realiza una consulta, el resultado sea siempre igual u homólogo con el transcurso del tiempo.
- **Integrar almacenamiento de datos y *Streaming Data*** En general, cuando se trabaja con procesamiento de datos, es importante guardar estados en el sistema, de tal manera que los datos entrantes vayan verificando, modificando o eliminado la información que se posea. En un operador que cuente palabras, es necesario soportar variables que guarden las estadísticas de la información entrante. Otro tema importante es la uniformidad de los datos, como se había presentando en el tópico anterior de *Streaming*, siempre se va a trabajar con datos heterogéneos, por lo que se requiere estandarizarlos para su procesamiento, de esta manera, no existirá una discordancia en la información procesada.
- **Garantizar la seguridad y disponibilidad de los datos** Este requerimiento está orientado en poseer mecanismos de *checkpoint*, técnica utilizada para respaldar el estado del operador cada cierto período de tiempo, y tolerancia a falla, por lo que en caso de existir alguna anomalía, pueda volver el sistema a estar disponible y sin perder una cantidad considerable de información, ya sea en las estadísticas o estados del sistema.
- **Partición y escalabilidad automática de las aplicaciones** Es importante también distribuir la carga entre distintos procesadores o máquinas, deseando idealmente una escalabilidad incremental, esto significa que el flujo de datos sea entregado a los distintos recursos que se posean y en caso de necesitar más recursos, incrementar lo que se poseen (Tanenbaum & van Steen, 2007). Si bien no sucede siempre, se espera que esto sea automático y transparente.

- **Procesamiento y respuesta instantánea** Cuando se plantea el uso de los SPS, se apuesta por un sistema que entregue respuestas en un tiempo lo más cercano al real, se hace necesario lidiar con las sobrecargas de flujos de datos que puedan generarse, las cuales afectan al rendimiento del sistema. Por lo tanto, se hace necesario una optimización con bajo *overhead*, esto quiere decir con bajo costo de implementación o recursos necesarios para su funcionamiento, aumentando la eficiencia y rendimiento del sistema.

Cada sistema de procesamiento de *streaming* está basado en un modelo de procesamiento en particular. Por ejemplo, S4 está basado en el modelo de procesamiento *push* (Neumeyer et al., 2010), y Storm en el modelo *pull* (Storm, 2014).

El primer modelo consiste en el envío de datos desde el operador. La ventaja de este modelo empleado por S4 radica en la abstracción en el envío de datos, sin embargo no asegura el procesamiento de estos, debido a que no existe un mensaje de respuesta al ser entregado al operador. En la Figura 2.3 se puede ver el Operador A como envía los datos al Operador B, donde en caso que esté procesando un dato el Operador B, éste lo guardará en cola. No existe algún mecanismo para asegurar que llegue efectivamente el dato, puede darse el caso que por falla de la red no se haya enviado u otro motivo, por lo que existe una abstracción en el envío de los datos.

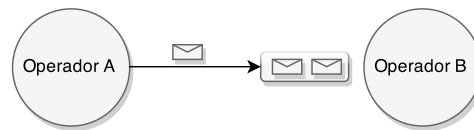


Figura 2.3: Modelo push.

En cambio, en el segundo modelo se basa en la petición de datos a un operador, por lo que son enviados solo si son requeridos. Si bien este modelo asegura procesamiento de los datos, genera una menor abstracción al programador, dado que en el primer modelo sólo se indica a que operador deben ir los datos, en cambio en el segundo se debe indicar quién lo envía y quién lo recibe. En la Figura 2.4 se puede ver que existen dos operadores, donde en la parte (a) se solicita por parte del Operador

B el envío de un dato para ser procesado, donde en la parte (b) el Operador A envía el dato para que posteriormente sea procesado por el Operador B.

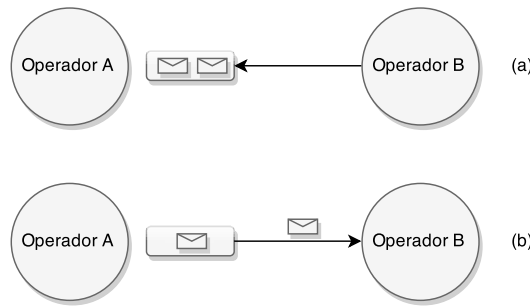


Figura 2.4: Modelo pull.

## 2.4 ELASTICIDAD

La propiedad de elasticidad en el área de *Cloud Computing* o *SPS*, está relacionado con la capacidad que el sistema tiene de adaptarse dinámicamente. Esto quiere decir que aumente o disminuya los recursos que se utilicen, para que funcione de manera eficiente.

En el caso de *Cloud Computing*, existen estudios que han trabajado con esta propiedad como (Gong et al., 2010; Nguyen et al., 2013; Lehrig et al., 2015), donde el sistema se comporta de forma elástica, determinando dinámicamente la cantidad de máquinas virtuales necesarias en el sistema. Por otra parte, en los SPS, existen trabajos como (Gedik et al., 2014; Ishii & Suzumura, 2011; Schneider et al., 2009; Madsen et al., 2014; Gulisano et al., 2012), en que el sistema de forma dinámica determina la cantidad de operadores necesarios para realizar una tarea en específico, como se ve representando en la Figura 2.5, donde la cantidad de operadores B cambia dinámicamente según el rendimiento del sistema.

Un ejemplo práctico de elasticidad es el supermercado, donde se debe considerar la cantidad de cajas necesarias para atender de manera eficiente los  $n$  clientes que van llegando en un período de tiempo. Si se estudia el período de la mañana, en general, es tiene un bajo flujo de personas que acude al supermercado, en comparación con la tarde, pero alto con la medianoche. Por lo tanto, en los

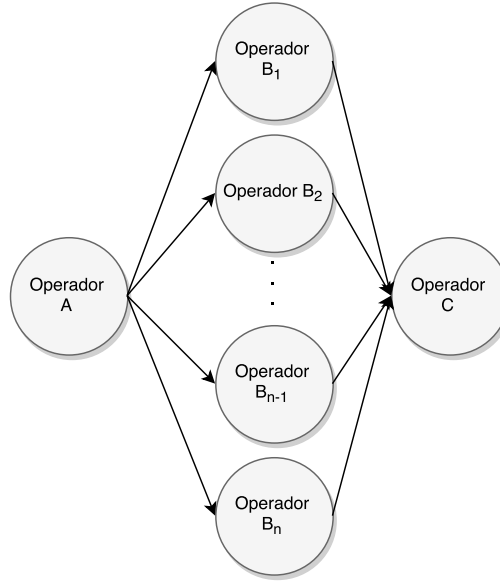


Figura 2.5: Elasticidad en un SPS.

horarios de la tarde es necesario poseer una mayor cantidad de cajas disponibles que en la tarde, disminuyendo la cantidad cuando el horario bordea la media noche, adaptándose de forma elástica la cantidad de cajas disponibles en el supermercado.

## 2.5 PROCESOS ESTOCÁSTICOS

Se define proceso estocástico como una colección de variables aleatorias  $X_t$ , con  $t \in T$ , las cuales están determinadas por algún comportamiento en el tiempo  $t$ . Esto significa que cada variable estará tratada de forma discreta en el tiempo, sin poseer un proceso determinístico entre sus variables, es decir, que las variables dependen de la historia (Taylor & Karlin, 2014).

Por lo tanto, se puede definir un estado como el posible comportamiento que puede tener una variable aleatoria en el sistema. Un ejemplo de esto es un modelo que contemple tres estados: estable, inestable y ocioso, y que según el valor de la variable aleatoria vaya cambiando de un estado a otro. Un caso de estudio utilizando el concepto de estados son las cadenas de Markov, las cuales consideran distintos estados que representan un comportamiento del sistema, habiendo una probabilidad de cambiar de un estado a otro (De Sapio, 1978).



### 2.5.1 Cadena de Markov

Sea  $X_t$  el valor de una variable aleatoria  $X$  en un tiempo  $t$ . El conjunto de todos los valores posibles para  $X$  se llama espacio de estado (Ching & Ng, 2006). La variable aleatoria es un proceso de Markov si las probabilidades de transición entre dos estados cualquiera de  $\Omega$  (definido como el universo de posibles estados), sólo depende del estado actual, como se denota en la Ecuación 2.1 y gráficamente en la Figura 2.6. Cabe destacar que este tipo de proceso es un caso específico de los procesos estocásticos.

$$P_r(X_{t+r} = S_j | X_0 = S_k; X_1 = S_l; \dots; X_t = S_i) = P_r(X_{t+1} = S_j | X_t = S_i) \quad (2.1)$$

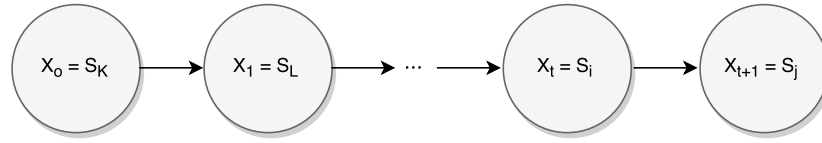


Figura 2.6: Proceso de Markov.

Una cadena de Markov es una secuencia de variables aleatorias generadas por un proceso de Markov, como se denota en la Ecuación 2.2.

$$(X_0, X_1, X_2, \dots, X_{n-1}, X_n) \quad (2.2)$$

La Ecuación 2.3 se define por sus probabilidades de transición. En la Figura 2.7 se muestra un ejemplo de la transición del estado  $i$  al estado  $j$ , dada la probabilidad  $P_{ij}$ .

$$P_{ij} = P_r(i \rightarrow j) = P_r(X_{t+1} = S_j | X_t = S_i) \quad (2.3)$$

En la Ecuación 2.4 se presenta una matriz de transición de finitos estados, donde la probabilidad de pasar de un estado a otro está determinado por una posición de la matriz, tomando en consideración que la suma de todas las transición de un estado debe ser igual a 1.

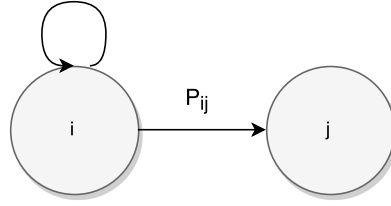


Figura 2.7: Cadena de Markov.

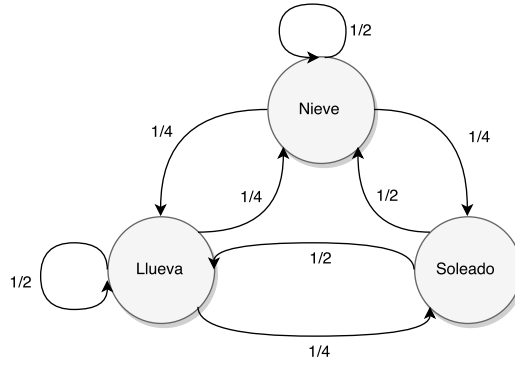


Figura 2.8: Ejemplo de cadena de Markov.

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,n} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n,1} & P_{n,2} & \cdots & P_{n,n} \end{bmatrix} \quad \sum_{j=1}^n P_{ij} = 1; \forall i \quad (2.4)$$

En la Figura 2.8 se muestra un ejemplo de una cadena de Markov simple, donde se analiza la probabilidad del clima de mañana dado el clima de hoy día. Como se puede observar, no se considera la historia del clima en la semana, sólo en el caso actual, lo cual es aplicado en los procesos estocásticos. Dada las probabilidades que transite de un clima a otro, se puede ver en la Ecuación 2.5 la matriz de transición resultante.

$$P = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \end{bmatrix} \quad (2.5)$$

Si se desea saber la probabilidad que la cadena esté en el estado  $S_i$  en el tiempo  $t + 1$ , está dada por la ecuación de Chapman-Kolmogórov (Papoulis, 1984):

$$\begin{aligned}
\Pi_i(t+1) &= P_r(X_{t+i} = S_i) \\
&= \sum_k P_r(X_{t+i} = S_i / X_t = S_k) P_r(X_t = S_k) \\
&= \sum_k P_r(X_{t+i} = S_i / X_t = S_k) \Pi_k(t)
\end{aligned} \tag{2.6}$$

En notación matricial:

$$\begin{aligned}
\Pi_{(t+1)} &= \Pi_{(t)} P \\
\begin{bmatrix} \Pi_1 & \Pi_2 & \Pi_3 \end{bmatrix}_{(t+1)} &= \begin{bmatrix} \Pi_1 & \Pi_2 & \Pi_3 \end{bmatrix}_{(t)} \begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,n} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n,1} & P_{n,2} & \cdots & P_{n,n} \end{bmatrix}
\end{aligned} \tag{2.7}$$

Usando recurrencia, se puede calcular la distribución estacionaria como se muestra la ecuación 2.8, la cual indica el comportamiento a futuro de la cadena de Markov, dado los estados y transiciones que éste posee.

$$\begin{aligned}
\Pi(t) &= \Pi(t-1)P \\
&= \Pi(t-2)P^2 \\
&= \Pi(0)P^t; \Pi(0) : \text{distribución inicial}
\end{aligned} \tag{2.8}$$

### 2.5.2 Trabajo relacionado

Existen modelos predictivos que están basados en modelos matemáticos, los cuales simulan el comportamiento del sistema, ya sea del flujo o de la carga de un operador, de tal manera que pueda predecir como será su estado en un tiempo futuro. En general, para poder realizar una predicción se analiza las variables deseadas en una ventana de tiempo, para posteriormente aplicar un modelo matemático que prediga

la variación del sistema en la próxima ventana de tiempo que se tiene estipulada.

Dentro de las aplicaciones que se han realizado con modelos predictivos, se encuentra PRESS (Gong et al., 2010). En este sistema orientado a *Cloud Computing* (Birman, 2012), analiza la cantidad de recursos disponibles, ya sea la memoria disponible o el uso promedio de CPU, en las máquinas virtuales que se dispone en el *Infrastructure-as-a-Service*. Para realizar la predicción del estado del sistema, se aplica cadenas de Markov, tomando sus estados como ventanas de tiempo en un determinado período. De esta manera, se analiza el estado del sistema en un tiempo en específico, para analizar si posee correlación con algún estado de la cadena de Markov, para posteriormente ver la transición de ese estado a otro y generar la matriz de transición. Posteriormente, con la ecuación de Chapman-Kolmogorov, se calcula la distribución estacionaria de la matriz estacionaria, de tal manera de saber en que estado estará en la próxima ventana de tiempo, para finalmente analizar si es necesario algún cambio en el sistema.

Dentro de la misma línea de modelos predictivos, existe el sistema AGILE (Nguyen et al., 2013) para *Cloud Computing*, que modifica las máquinas virtuales de forma elástica en un *Infrastructure-as-a-Service*. Lo que se realiza en este trabajo es aplicar la transformada de Fourier (Falk et al., 2012) a la carga de CPU en una ventana de tiempo determinada, donde la función resultante se analiza con distintas frecuencias, de tal manera de solicitar la predicción de la próxima ventana de tiempo a cada una de las funciones creadas. De esta manera, se sintetizan todas predicciones realizadas por cada función, para analizar el comportamiento del sistema en la próxima ventana de tiempo, y ver si es necesario aumentar o disminuir recursos de éste.

## 2.6 TEORÍA DE COLAS

La teoría de colas se centra en el estudio matemático de las colas existentes en un sistema, cuyo caso de estudio era el desbordamiento de peticiones por parte del cliente al servidor (Breuer & Baum, 2005). En la Figura 2.9 se muestra un ejemplo

de un sistema basado en teoría de colas, donde existe  $n$  productores que envían cierto flujo de datos a los  $m$  servidores disponibles, y en caso de no estar disponibles, se genera una cola de espera en el sistema.

- **Productor** es quién provee la fuente de entrada para el servidor, de tal manera que procese según la necesidad que se posea.
- **Cola** o línea de espera, la cual está encargada de almacenar la información emanada por el productor en caso que los servidores estén ocupados, para que posteriormente sean procesados.
- **Servidor** es quién procesa la información disponible en la cola, de tal manera que entre una fuente de salida con los datos o información deseada.

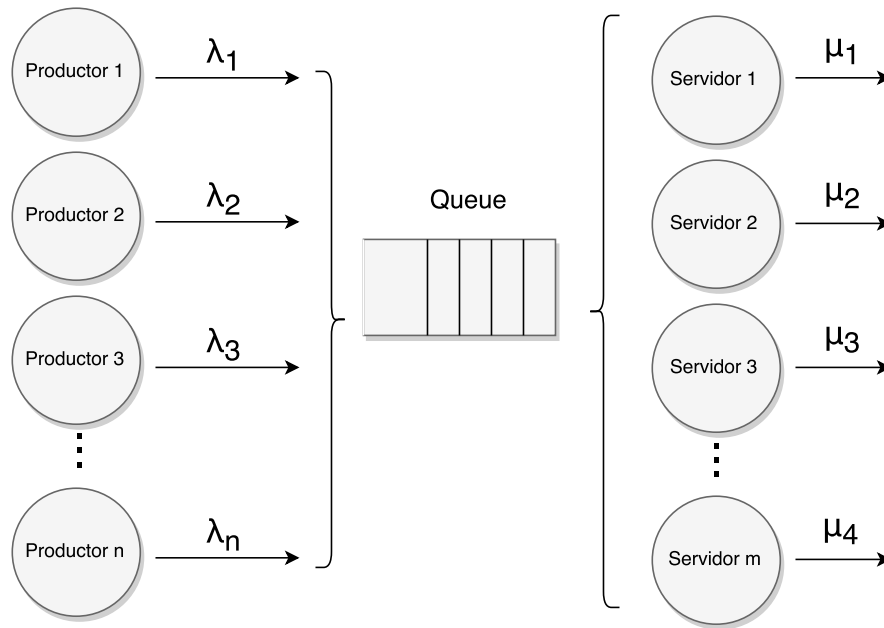


Figura 2.9: Ejemplo de un sistema basado en teoría de colas.

Además de esto, se tienen ciertos componentes importantes en el sistemas, definidos a continuación:

- **Tasa de llegada**, denotado  $\lambda$ , es la cantidad de datos, eventos o información que van llegando por un determinado período de tiempo, la cual está determinada por los productores que existan en el sistema.

- **Tasa de procesamiento**, denotado  $\mu$ , es la cantidad de datos, eventos o información que salen del sistema, producto del servicio provisto por cada servidor.
- **Tasa de rendimiento**, denotado  $\rho$ , es el porcentaje de utilización del sistema, donde  $\rho = \frac{\lambda}{\mu}$ , siendo un sistema estable si  $\rho < 1$ , dado que la capacidad de procesamiento es mayor que la tasa de llegada.
- **Disciplina de la cola** significa el método utilizado para extraer los datos encolados en el sistema, para esto puede aplicarse los métodos *FIFO*, *LIFO*, *RSS*, entre otros.

Este tipo de modelos se puede aplicar en los SPS, debido que la fuente de datos es el productor y cada operador es un servidor del sistema. Por lo que existe un problema interesante a analizar, dado el dinamismo de los datos a procesar, pudiendo generarse sobrecargas en algún operador. Esto se produce debido a que la tasa de procesamiento es menor a la tasa de llegada, creando colas en el sistema. Por ejemplo, si se posee una tasa de llegada  $\lambda$  y una tasa de servicio  $\mu$ , donde  $\mu < \lambda$ , se tendrá un sistema inestable, debido que se procesa más lento de lo que llegan los datos. Como existen colas, es necesario un aumento del rendimiento del sistema, debido que  $\rho > 1$ , donde se define  $\rho = \frac{\lambda}{s\mu}$ , siendo  $s$  la cantidad de servicios disponibles.

## CAPÍTULO 3. BALANCE DE CARGA EN SPS

### 3.1 PERSPECTIVAS DE BALANCE DE CARGA

Dentro de la literatura se han encontrado distintas perspectivas al problema de balance de carga en un SPS (Sistema de Procesamiento de *Streaming*), las cuales consideran los recursos físicos o lógicos como problemas de la sobrecarga del sistema.

#### 3.1.1 Recursos físicos

En esta perspectiva se toma en consideración la sobrecarga del sistema dado las limitantes físicas que éste posea, ya sea por condiciones de los recursos disponibles o por el ambiente de desarrollo. Para esto, se consideran distintos parámetros como umbrales, los cuales si son sobrepasados debe aplicarse alguna estrategia para aliviar la carga del sistema. Estos umbrales pueden ser el nivel *Service Level Objective* (SLO) (Sturm et al., 2000), porcentaje de CPU utilizada o disponibilidad de la memoria (Dong & Akl, 2006).

Una de las soluciones con la perspectiva anterior es la de Borealis (Xing et al., 2005), donde considera la cantidad de carga de los nodos en ventanas de tiempo determinadas, las cuales serán manejadas por un coordinador centralizado. Este coordinador se encarga de analizar los recursos del sistema, y en caso de sobrepase el umbral propuesto, se deberán migrar los operadores que estén en ese nodo, para luego ser enviados a otro nodo candidato con menor cantidad de carga. Para elegir al nodo candidato, se realiza un análisis de la cantidad de correlación que existe entre el operador y el nodo candidato, de esta manera, no necesariamente va a ser enviado a otro nodo con menor sobrecarga, sino también a uno que posea menor cantidad de mensajes. Dentro de los problemas que pueden existir en el sistema es la conexión entre los distintos nodos, por lo que para las pruebas se considera un buen ancho de banda, de tal manera que aparente una red sin limitaciones de recursos.

Otra de las soluciones que se han propuesto es lo realizado por Flood (Alves et al., 2010), la cual es un DPS (*Distributed data stream processing*) que considera ciertos factores físicos para agregar o eliminar máquinas virtuales que se provee del *Infrastructure-as-Service*, como Amazon EC2. Para esto, se posee un administrador que considera las estadísticas en tiempo de ejecución como la cantidad de CPU utilizada, latencia o memoria disponible, las cuales considera para ver en que rango está de los umbrales establecidos, y posteriormente agregar o eliminar recursos de manera elástica.

### 3.1.2 Recursos lógicos

A diferencia de la física, en esta perspectiva se consideran los componentes lógicos del sistema, como la carga de un operador. Las distintas soluciones que se presentan, analizan componentes como el flujo de datos o el tamaño de la cola de un operador, tomando esos parámetros como umbrales en los algoritmos implementados para realizar mejoras en el sistema.

Dado esta perspectiva, se han presentando dos tipos de enfoques: el estático y el dinámico (Gupta & Bepari, 1999). El primer enfoque está centrado en un modelo definido y fijo antes de la inicialización del sistema, sin considerar el estado del mismo. En cambio, el segundo enfoque está basado en un modelo que analizará el sistema según su estado en el transcurso de su ejecución.

### 3.1.3 Enfoque estático

Este enfoque se ha implementando en distintos sistemas de procesamiento de *stream*, donde no se depende del estado del sistema (Storm, 2014; S4, 2014). De esta manera, no existe una interrupción en la ejecución o un cambio debido al estado del sistema (Casavant & Kuhl, 1988). Por lo tanto, no se considera variables como la carga o cola del operador, sólo se aplican técnicas que administren el flujo de los datos en el sistema.



*Storm* utiliza distintas técnicas de distribución de las tuplas en los operadores según la política que se desee, todas tomando el enfoque estático (Storm, 2014). Dentro de las políticas que existen están *Shuffle grouping*, *Fields grouping*, *Partial Key grouping*, *All grouping*, *Global grouping*, *None grouping*, *Direct grouping* y *Local grouping*.

La política de *Shuffle grouping* se enfoca en distribuir las tuplas de forma homogéneas en los  $n$  operadores que se encuentren en el grafo, utilizando la planificación *Round-Robin* (Brucker, 2004), de esta manera la cantidad de tuplas se distribuye de forma homogénea en el sistema. Una de las principales fallas es que la tasa de procesamiento de las tuplas no siempre es la misma, por lo tanto puede existir una sobrecarga en un operador que le llegue mayor cantidad de tuplas con un mayor tiempo de procesamiento. Otra de las políticas muy utilizadas es *Fields grouping*, la cual determina ciertas llaves a un operador determinado, es decir, si la llave fueran palabras desde una letra hasta otra letra serán procesadas por cierto operador. Si bien genera un determinismo en el procesamiento de las llaves, puede existir una sobrecarga de un operador, debido que una llave se repite con mayor frecuencia que otras (Leibiusky et al., 2012).

Por otra parte, se encuentra el funcionamiento de S4, cuya política es similar a la de *Fields grouping* de Storm, la diferencia es que un operador no le corresponde un conjunto de llaves, sino que posee una llave única. Esto quiere decir que cada llave se le asigna un operador, y en caso de no existir un operador para el valor de esa llave, se creará un nuevo operador para esa llave. Debido a la infinidad de combinaciones en la llave, S4 recomienda aplicar una función *hash* (Rogaway & Shrimpton, 2004), de esta manera el valor de la función determina el operador que procesará el dato y disminuirá la cantidad de operadores que deben estar disponibles. Esta técnica provee dinamismo en la cantidad de operadores en el sistema, pero al igual que la *Fields grouping* puede sobrecargar un operador, debido que una llave posee mayor frecuencia que las otras.

Una ventaja del enfoque estático es el bajo costo de la implementación de los métodos, lo cual es beneficioso para sistemas con bajos recursos. Por otra parte, una desventaja existente es la sobrecarga de un nodo u operador, debido que

no asegura que la cantidad de flujo sea repartido de forma homogénea. Si bien, no es una solución óptima, es un buen complemento para un modelo con el enfoque dinámico.

#### 3.1.4 Enfoque dinámico

Este enfoque está basado en el estado del sistema, siendo esto el parámetro para optimizar su rendimiento (Casavant & Kuhl, 1988). Esto significa que si el sistema posee una anomalía, como una sobrecarga o latencia entre nodos, es necesario realizar un cambio en el sistema con el fin de solucionar estos problemas. En este contexto se consideran dos modelos: reactivo y predictivo.

**Reactivo** Este modelo está basado en la detección de sobrecargas en el sistema a través de un monitor (Gulisano et al., 2012), el cual recibe periódicamente las variables de cada uno los operadores, y en caso que sobrepase un umbral, se aplica una técnica para aumentar el rendimiento bajo una métrica dada. El umbral puede estar basado en el tiempo de procesamiento, el tamaño de la cola u otra variable del operador (Bhuvanagiri et al., 2006). Por ejemplo, para realizar una optimización en el rendimiento general del sistema, se considera la tasa de procesamiento de cada operador, por lo que en caso de existir congestión en un operador, se procede a realizar una paralelización del operador, de tal manera que exista un operador adicional que pueda recibir un flujo de datos y realizar la misma operación que el operador sobrecargado (Schneider et al., 2009).

Si bien estas soluciones en su mayoría son eficientes y poseen buen rendimiento, uno de los principales problemas es que no analiza el comportamiento a futuro, debido que sólo analiza y resuelve la situación en el momento. Otro problema son los falsos positivos, debido que puede ser que en un momento exista un *pick* de tráfico, pero esto era sólo un caso particular de un tiempo determinado, por lo que no era necesaria la mejora en el sistema.

**Predictivo** Este modelo está basado en modelos matemáticos que calculan o estiman el comportamiento a futuro del sistema, dada cierta información que se posee del sistema, como flujo entrante o carga de la CPU. Si bien no existen modelos predictivos para SPS, si los existen en otras áreas, como se explicó anteriormente en la subsección 2.5.2.

## 3.2 TÉCNICAS DE BALANCE DE CARGA

Existen distintas técnicas de balance de carga que utilizan alguno de los dos modelos presentados anteriormente, las cuales están enfocados a mejorar el rendimiento del sistema en caso de existir una sobrecarga (Hirzel et al., 2013). Dentro de las técnicas existentes se encuentran la planificación determinista (Xu et al., 2014; Dong et al., 2007), *load shedding* (Sheu & Chi, 2009), migración (Xing et al., 2005) y fisión (Gulisano et al., 2012; Ishii & Suzumura, 2011; Gedik et al., 2014; Fernandez et al., 2013), si bien existen más, sólo se trataron estas porque se consideran las relevantes.

### 3.2.1 Planificación determinista

La planificación determinista se centra en los conocimientos *a priori* del sistema, esto significa que se consideran las variables del entorno que se poseen y respecto a esto se toma una decisión de como debe actuar el sistema.

En el área de *Stream processing*, se han realizado diferentes análisis de la estimación de frecuencia de *data stream* en el sistema. Para poder realizar esto, se han considerado modelos matemáticos, tomando ventanas de tiempo de la frecuencia predicha y la real, para posteriormente generar con los datos una función que represente la frecuencia estimada del operador (Ganguly, 2009). Pero no sólo se han considerado modelos matemáticos, sino también algoritmos que determinan la frecuencia del sistema dado el flujo de datos que se podría poseer (Bhuvanagiri et al.,

2006).

En otras áreas, como red de sensores, se utiliza esta técnica en el envío de estadísticas de dispositivos móviles, los cuales manejan información *a priori* de donde están los sensores, de tal manera de determinar según la intensidad de la frecuencia, localización o clima, a dónde debe enviar la señal para que se recolecte la información correspondiente (Dong et al., 2007).

Una de las limitaciones es que si bien realiza una predicción determinista de la frecuencia, no necesariamente es correcta a futuro. Esto se debe a qué puede analizarse respecto al promedio, pero pueden surgir procesos anómalos en el transcurso de la ejecución que generarán una sobrecarga en el sistema. Por lo tanto, la estimación al realizarse *a priori*, sólo podrá considerar el inicio del sistema o ventanas de tiempo, por lo que puede existir un porcentaje de error considerable. Por otra parte, se considera que posee mejor rendimiento esta técnica si es que la frecuencia o función analizada es estacionaria (Karp et al., 2003).

### 3.2.2 Load Shedding

En los SPS también se utiliza la técnica de *load shedding*, que consiste en descartar eventos del sistema en caso de existir un comportamiento anómalo, ya sea un máximo en el tamaño de la cola u otro factor. En la Figura 3.1 se puede ver que existe un operador A, el cual le llega cierta cantidad de datos en un período de tiempo, debido a la cola que existe por parte del sistema, se considera utilizar un operador denominado *Shedding*, que en caso de existir un flujo de datos mayor al umbral propuesto, va a descartar los eventos que excedan el umbral. Por ejemplo, en la transmisión de *video streaming*, al enviar el flujo de información existe un administrador que está analizando el contenido a procesar, por lo que en caso de llegar datos de baja calidad, serán descartados por éste. De esta manera, al existir menor cantidad de datos que procesar, el sistema posee un mejor rendimiento, además de tener una mejor calidad en la visualización de los videos, dado que en su mayoría se procesan datos de alta calidad (Sheu & Chi, 2009).

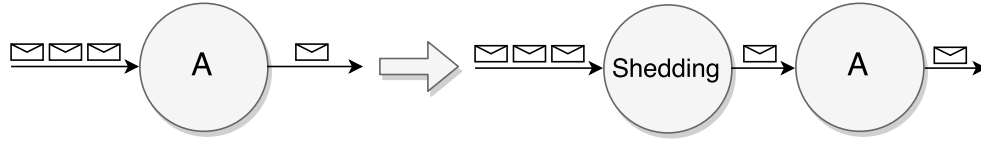


Figura 3.1: Load shedding en un SPS.

En el mundo de los SPS, varios poseen este tipo de estrategia, como por ejemplo S4 (S4, 2014), doonde se establece una cota superior de eventos en cola, y en caso que su cola sea igual al límite establecido, los eventos entrantes serán descartados. Otro sistema que aplica esta técnica es Aurora (Abadi et al., 2003), el cual se basa en procesamiento de datos por ventanas de tiempo, por lo que en caso de existir una ventana de tiempo con una mayor cantidad de eventos de lo estipulado, se descarta el exceso de eventos.

Si bien esta técnica es simple y de bajo costo, siendo pensada para la disminución rápida de carga, existe una baja en la precisión y fiabilidad del procesamiento de los datos. Por ejemplo, en el caso de la transferencia de video no es trascendental, dado que son pocos los pixeles perdidos, pero en una recopilación y análisis de estadísticas, esto dará una menor precisión de los datos procesados por el sistema, dado que puede perderse información que indique comportamientos de los datos estudiados.

### 3.2.3 Migración

La técnica de migración está basada en el traspaso de un operador de un nodo a otro, según el estado del sistema. En la Figura 3.2 se puede apreciar dos nodos, los cuales poseen tres y dos operadores respectivamente, pero debido a una sobrecarga del nodo 1 es que se realiza una migración de un operador al nodo 2, ya que este se encuentra con menor carga, de esta manera, se reparten homogéneamente la carga dada la carga que exigen los operadores que posee cada uno de los nodos. Si bien no existe alguna implementación que utilice los recursos lógicos, si existe una que utiliza los recursos físicos como es el caso de Borealis, el cual fue explicado anteriormente (Xing et al., 2005). Una de las principales críticas que se realiza a

esta técnica, es la transferencia de datos, por lo que existe una menor tolerancia a fallos, por lo que se propone el uso de *buffer* que tengan respaldos de la información, aumentando los costos del sistema (Pittau et al., 2007).

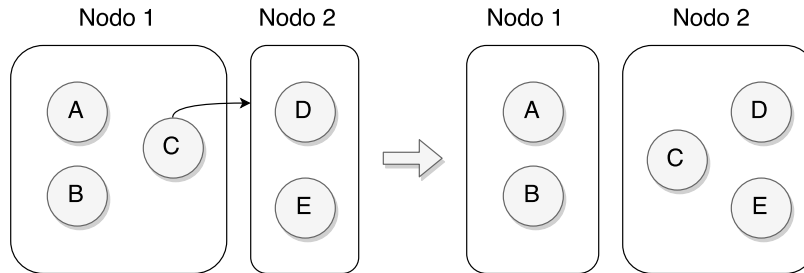


Figura 3.2: Técnica de migración en un SPS.

### 3.2.4 Fisión

Otra técnica utilizada en el balance de carga es la fisión, o también llamada replicación, particionamiento o paralelismo, la cual consiste en crear una réplica paralela del operador, sin perder el funcionamiento y estado, en caso de existir una sobrecarga en el operador. En Figura 3.3 se puede ver que existe un operador A, el cual en primera instancia tiene un flujo de entrada  $q_1$  y flujo de salida  $q_2$ , pero debido a la sobrecarga del operador A, se crea dos operadores extras, los cuales pasarán por un *Split* y posteriormente por un *Merge*, que tendrán como función distribuir y juntar la información respectivamente. En ciertos SPS se posee el planteamiento que el *split* y el *merge* son operadores que deben ser realizados por el programador, y no de forma automática por el sistema, como S4 o Storm. Una de las características que se posee de esta técnica es la elasticidad del sistema, donde aumenta o disminuye la cantidad de operadores según la necesidad del sistema.

Una aplicación que aplica la técnica de fusión según el enfoque estático, es la paralelización de tareas de Storm (Leibiusky et al., 2012), donde se indica la cantidad de operadores necesarios para realizar una tarea paralelamente. De esta manera, existe un proceso que está encargado de la tarea deseada, y  $n$  hebras por la cantidad de operadores que se deseen.

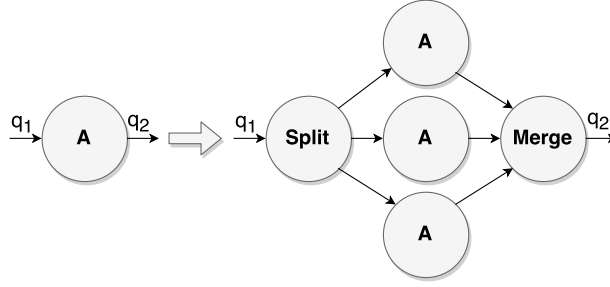


Figura 3.3: Técnica de fisión en un SPS.

Otro sistema que utilizan esta técnica, mediante el enfoque dinámico, es *StreamCloud* (Gulisano et al., 2012), donde según la cantidad de consultas realizadas al sistema, se aumenta o disminuye la cantidad de operadores que cumplen las tareas que se solicitan. Como se había mencionado anteriormente, existe un problema con la variable de distribución y unión de la información al replicar los operadores, este sistema propone que pueda resolver ciertas consultas el sistema y que de forma automática pueda realizar el proceso de *split* y *merge*, de tal manera de no tener problemas con los operadores con estado, como lo son los contadores y algoritmos de ordenamiento. Una de las características principales de este sistema, es aplicar el concepto de elasticidad, que aumenta y disminuye la cantidad de operadores según lo requerido por el sistema. Otros trabajos como (Gedik et al., 2014; Schneider et al., 2009) también aplican este método, y paralelizan las tareas de forma elástica, y con parámetros similares, sólo que su implementación es distinta.

El último trabajo a analizar según esta técnica es lo realizado por Fernández (Fernandez et al., 2013), donde aplican fisión en el caso que exista un cuello de botella en un operador. Para la detección de estas situaciones, se posee un monitor, el cual está consultando en un período de tiempo corto el estado de cada uno de los operadores. De esta manera, se puede ver en cada operador si sobrepasa el umbral propuesto, que era en este caso la utilización de la CPU, se procede a replicar el operador sobrecargado. En la Figura 3.4 se puede ver el caso de un simple sistema, en el cual el operador  $u$  está enviando un flujo de datos al operador  $o$ , hasta que en cierto momento el operador  $o$  posee un cuello de botella y debe replicar, hasta que en la parte (c) denota que convergió el operador  $o$ , es decir llegó a la cantidad de réplicas necesarios, y ya no es necesario replicar más, pero con el transcurso del

tiempo si fue necesario en el operador  $u$ , por lo que se aplica la misma lógica descrita anteriormente con el operador  $o$ .

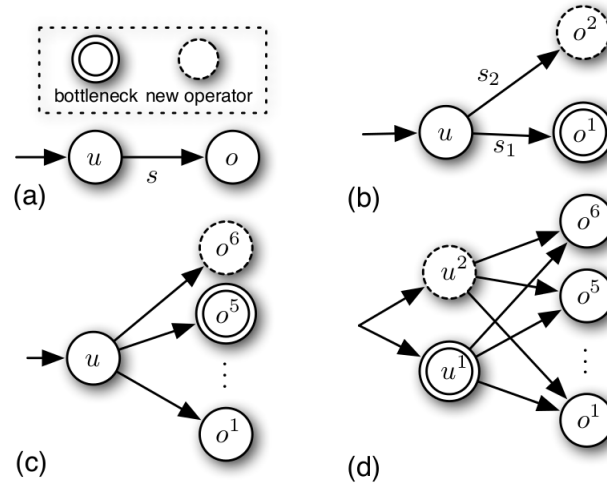


Figura 3.4: Ejemplo de replicación de los operadores (Fernandez et al., 2013).



## **CAPÍTULO 4. DISEÑO DEL SISTEMA DE DISTRIBUCIÓN DE CARGA**

Como se había mencionado en la subsección 1.2, el problema de la sobrecarga en un SPS está dado por la inflexibilidad existente en el grafo diseñado. Esto quiere decir, que en el momento que el sistema está funcionando, no existen un cambio en la cantidad de recursos necesarios, por lo que se vuelve ineficiente, debido a la sobrecargada en el sistema.

De ser así, es necesario un sistema que pueda proveer dinamismo en su estructura, así como también en el análisis de la carga tanto en el momento como a futuro, de tal manera que se complementen. Con estas condiciones, se diseñó un sistema de bajo costo que optimice su rendimiento, sin generar interrupciones en la ejecución del sistema.

### **4.1 ANÁLISIS DEL SISTEMA DE DISTRIBUCIÓN DE CARGA**

Dentro del análisis realizado en la arquitectura del sistema implementando, se consideró una perspectiva en base a los recursos lógicos según el enfoque dinámico, definido en las subsección 3.1.2 y 3.1.4 respectivamente, para el balance de carga de SPS. Esto debido que el trabajo presentando no analizó el comportamiento que tenga cada uno de los nodos del sistema, sino que se analizó el rendimiento que poseía cada uno de los operadores del grafo diseñado, siendo un problema de carácter lógico y no físico.

Respecto al estudio de las distintas técnicas implementadas, era necesario utilizar una que no tuvieran desventajas en cuanto a la pérdida de datos, inadaptabilidad con el tiempo y costo de implementación. Por lo tanto, se consideró que la mejor opción era utilizar la técnica de fisión, utilizando el mismo modelo de replicación que Fernández (Fernandez et al., 2013), donde según una sobrecarga en el operador era necesario generar una réplica de ese operador. Dentro de las hipótesis planteadas, se pensaba que el costo de un operador iba a ser menor

a la formación de las colas de datos en el sistema, lo cual podría variar según la arquitectura del SPS implementando.

En la Figura 4.1 se muestra un ejemplo de la replicación propuesta, donde en la parte (a) se presentan tres operadores, donde en uno de ellos (operador B) existe una sobrecarga, por lo que es necesario replicar el operador. En la parte (b) se realiza la replicación, pero todavía persiste la sobrecarga en el operador, por lo que se vuelve a realizar el mismo procedimiento, hasta que finalmente converge a la cantidad óptima de réplicas deseadas en el sistema en el período de tiempo analizado, como se muestra en la parte (c).

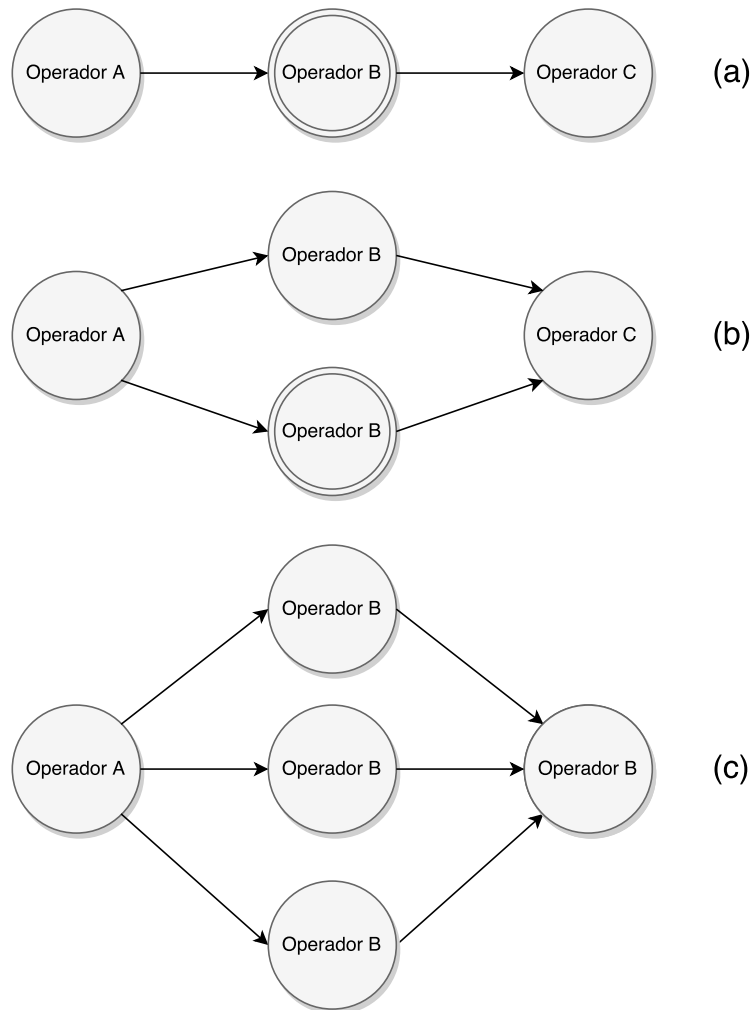


Figura 4.1: Ejemplo de replicación del sistema propuesto.

Para la detección de sobrecarga es necesario contar con un umbral que determine cuando está o no sobrecargado un operador, por lo que se utilizaron

conceptos de teoría de colas (Bose, 2013) para el análisis de esto. Como los SPS están orientados a grados, se puede obtener tanto la tasa de llegada ( $\lambda$ ) como la tasa de servicio ( $\mu$ ) de cada uno de los operadores, como se ve representando en la Figura 4.2, donde la tasa de procesamiento de un operador influye directamente en la tasa de llegada del siguiente operador en el grafo. Al utilizar estos conceptos, se cálculo la tasa de rendimiento ( $\rho$ ), la cual está definida por la tasa de llegada, de procesamiento y la cantidad de réplicas del operador ( $\rho = \frac{\lambda}{\mu\rho}$ ), cuyo valor representa el factor de utilización del sistema, donde se define un sistema estable si y sólo si  $\rho < 1$ .

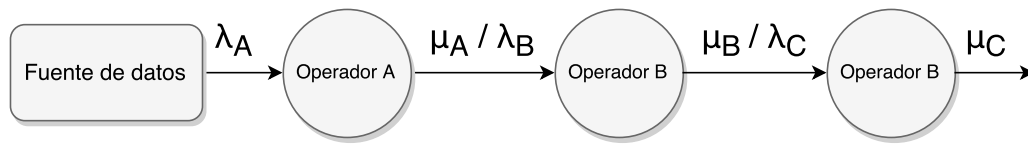


Figura 4.2: Enfoque de un SPS con conceptos de teoría de colas.

Tomando en consideración el tipo de enfoque en el algoritmo de balance de carga y la elasticidad que se pretendía por parte del sistema, es que se trataron tres posibles estados en el sistema: ocioso, estable e inestable. El primer estado corresponde a un exceso en la cantidad de recursos necesarios. El segundo está definido por el rendimiento óptimo del sistema. Y por último, el tercero hace referencia a un sistema sobrecargado, donde es necesario mayor cantidad de recursos por parte de éste. Definido los posibles estados de cada operador, es que se tomó esto como base para el análisis y predicción de la carga en el sistema de distribución de carga.

Para el sistema propuesto, se consideraron dos tipos de algoritmos: predictivo, enfocado en el futuro y la historia del operador, y reactivo, analizando el comportamiento del momento. Esto fue diseñado con el fin de analizar dos factores, los *peak* existentes en la historia del operador, dado el algoritmo predictivo, y otro que analice el comportamiento en el momento, haciendo uso del algoritmo reactivo, de tal manera de solucionar las anomalías que no son detectadas con la predicción.

Es importante denotar que dependiendo del tipo de caso es que un tipo de algoritmo va a funcionar mejor, por ejemplo si se posee una tasa de llegada dado una función exponencial, es necesario aumentar la cantidad de réplicas a medida que

van aumentando la cantidad de réplicas en el sistema, por lo que en el predictivo podrá detectar este tipo de casos y aumentar la cantidad réplicas dado este patrón. Pero en el caso que no exista un patrón en la función, existe el algoritmo reactivo que analiza la situación en el momento para ver si es necesario o no cambiar la cantidad réplicas.

Como se diseñaron dos tipos de algoritmos que se complementaran, es necesario considerar un algoritmo que administre cual de los dos algoritmos se va a utilizar según el período analizado, como también la cantidad de réplicas que deben crearse o eliminar según el resultado del algoritmo utilizado.

Dado lo anterior, se diseño un sistema de distribución de carga con cuatro componentes: monitor de carga, analizador de carga, predictor de carga y administrador de réplicas, que se pueden apreciar en la Figura 4.3.

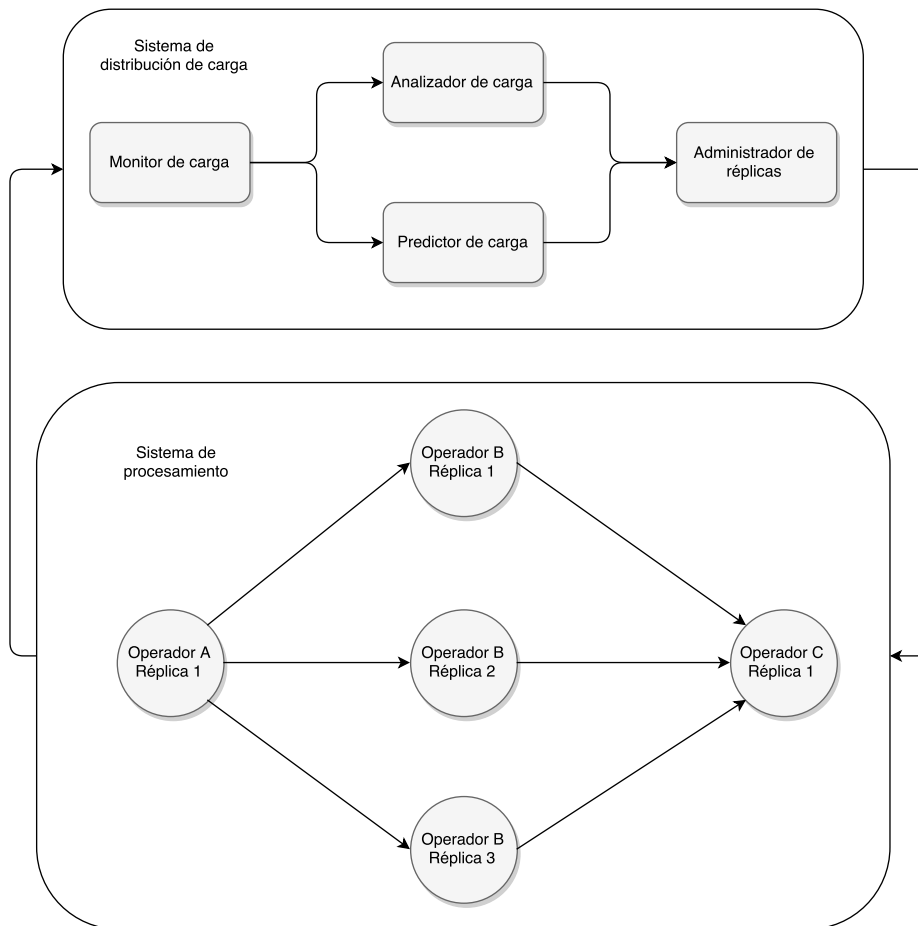


Figura 4.3: Estructura del sistema de distribución de carga.

**Monitor de carga** Está encargado de estar enviando las estadísticas al sistema, ya sea para recolectar las estadísticas para el algoritmo reactivo como el historial para el predictivo.

**Analizador de carga** Analiza la cantidad de carga de un operador en un período de tiempo determinado, y respecto a esto se indica el estado del operador. Para esto, se consideró la tasa de rendimiento del operador, y según el valor que posea se determinará el estado, el cual podía ser ocioso, estable o inestable.

**Predictor de carga** Analiza la historia de un operador en una ventana de tiempo determinada, utilizando como muestra la tasa de rendimiento del operador, para posteriormente realizar una cadena de Markov según los posibles estados del sistema. Posteriormente, para la predicción de la carga del operador, se calculó la distribución estacionaria (Papoulis, 1984), el cual va a entregar la probabilidad que el operador se encuentre en cada uno de los posibles estados.

**Administrador de réplicas** Se encarga de determinar cual es el algoritmo a utilizar en cada período de tiempo, ya sea reactivo o predictivo, y la administración de las réplicas dado el resultado del algoritmo utilizado.

## 4.2 RECOLECCIÓN DE LOS DATOS

Como se había mencionado anteriormente, el monitor de carga está encargado de recolectar los datos necesarios para el funcionamiento del sistema de distribución de carga, tanto el historial para el algoritmo predictivo, como la tasa de rendimiento para el algoritmo reactivo. Para esto se consideraron ventanas de tiempo de un segundo para la recolección de cada muestra para el análisis del historial, y cinco segundos para el análisis reactivo del operador.

La recolección de muestras para el historial, se considero un segundo debido que se consideraba un período estable para determinar el comportamiento del operador. Con esto, se iba a poseer una muestra de cien datos cada vez que se realice

el algoritmo predictivo, debido que cada cien segundo se va a ejecutar. La cantidad de muestras fue determinado según la literatura, debido que se consideraba un número apropiado para realizar una predicción del operador (Ching & Ng, 2006), de tal manera de no existir una deficiencia en la cantidad de muestras para la predicción deseada.

Por otra parte, para la obtención de muestras para el algoritmo reactivo, se consideraron muestras obtenidas en períodos de cinco segundo. La muestra corresponderá a la tasa de rendimiento del operador en ese período, la cual es utilizada por el algoritmo reactivo para determinar el estado del operador según los umbrales propuestos. Dentro de las consideraciones realizadas para la recolección de datos para el algoritmo reactivo, fue considerar que la tasa de servicio ( $\mu$ ) es homogénea con el transcurso del tiempo, debido que los datos procesados lo son, por lo tanto, se considera el valor más alto y ese es considerado en el procesamiento de los datos.

Cabe destacar que cuando el algoritmo predictivo se ejecuta, no es necesario la recolección de los datos del período, debido que el algoritmo reactivo no será utilizado y sólo considerado datos del momento. Sólo la recolección del historial es realizada en todo momento, dado que éstas son guardadas para posteriormente ser analizadas por el algoritmo predictivo.

### 4.3 ALGORITMO REACTIVO

El diseño del algoritmo reactivo se basó en el análisis del estado del operador en un período determinado, siendo definido su estado por una variable del operador, el cual dependerá del rango en que se encuentre dado los límites que se poseen. En este caso, se analizó según la tasa de rendimiento ( $\rho$ ), donde el estado del operador dependerá del valor que éste posea según los límites establecidos.

En el Algoritmo 4.1 se puede ver el análisis del estado de un operador según su tasa de rendimiento; en el caso que sea mayor a 1, su estado es inestable, menor a 0.5, significa que está en estado ocioso, y sino, significa que está estable.

Estos datos posteriormente serán considerados por el administrador de réplicas, el cual analiza el comportamiento que debe tener el sistema según lo indicado por el algoritmo.

---

**Algoritmo 4.1:** Algoritmo reactivo del sistema de distribución de carga.

---

**Entrada:** Tasa de procesamiento  $\rho$  del operador  $\phi$ .

**Salida:** Estado del operador, donde -1 significa estado ocioso, 0 estable y 1 inestable.

```

1: if  $\rho_\phi > 1$  then
2:   return 1
3: else if  $\rho_\phi < 0.5$  then
4:   return -1
5: else
6:   return 0
7: end if

```

---

En la Figura 4.4 se puede analizar el estado del operador según la tasa de procesamiento de forma visual. En los primeros segundos la tasa del operador es mayor al límite superior, lo cual indica que el sistema es inestable, es decir, el operador posee sobrecarga. Con el transcurso del tiempo, la tasa de rendimiento disminuye, ya sea por una optimización o disminución de la tasa de llegada, por lo que ahora el operador ya no se encuentra sobrecargado, sino se encuentra entre el límite inferior y superior, cuyo rango define al operador como un sistema estable.

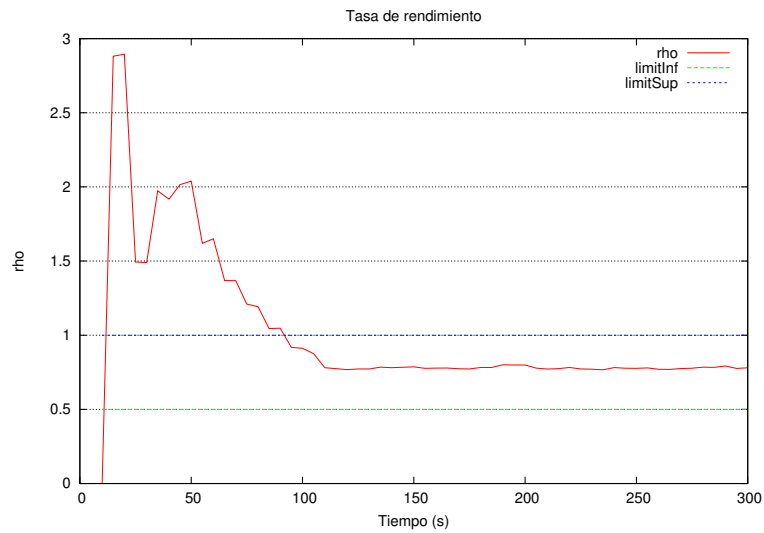


Figura 4.4: Comportamiento de la tasa de procesamiento de un operador.

#### 4.4 ALGORITMO PREDICTIVO

Para la confección del algoritmo predictivo se realizó un análisis según las cadenas de Markov (Ching & Ng, 2006), por lo que se tuvieron que seguir las siguientes condiciones:

- Definir muestras en tiempos discretos, las cuales cambiaran con el tiempo según un proceso estocástico. Las muestras se definieron como la tasa de procesamiento del operador, la cual dependiendo del valor que poseyera, iba a otorgar un estado al operador.
- Determinar los estados finitos que se van a utilizar para la conformación de la cadena, que en este caso sería los estados que puede encontrar el operador: ocioso, estable o inestable.
- Una cantidad representativa de muestras para la construcción de la cadena de Markov en el período analizado. Estas muestras serán independientes de un período y otro, por lo que los valores de la cadena de Markov irán cambiando en cada período de tiempo. Para la implementación del algoritmo, se consideraron cien muestras por cada período, cuyos intervalos eran de cien segundos.

Tomando las bases anteriores, se diseñó una cadena de Markov en base a tres posibles estados: ocioso, estable e inestable, como se demuestra en la Figura 4.5. Cada uno de los estados posee una probabilidad de transición hacia algún estado, cuyas probabilidades están definidas por las muestras obtenidas en el período de tiempo analizado.

Por lo tanto, para cada operador se construirá una cadena de Markov según el historial obtenido en la ventana de tiempo. Para la conformación de la cadena de Markov se consideraron las muestras de la historia, por lo que la transición de una muestra a otra presentaba una transición, las cuales dieron origen a la matriz de transición. En el Anexo A se puede ver el algoritmo que se empleó para construir la matriz de transición. En la ecuación 4.1 se muestra la matriz de transición que se obtiene de la cadena de Markov de la Figura 4.5.



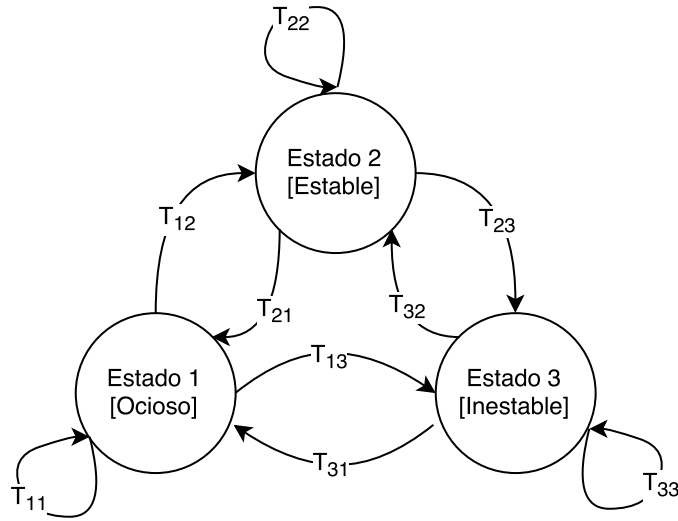


Figura 4.5: Cadena de Markov dado el modelo propuesto del sistema.

$$P = \begin{bmatrix} T_{1,1} & T_{1,2} & T_{1,3} \\ T_{2,1} & T_{2,2} & T_{2,3} \\ T_{3,1} & T_{3,2} & T_{3,3} \end{bmatrix} \quad (4.1)$$

Obtenida la matriz de transición se puede calcular la distribución estacionaria de la cadena de Markov, la cual indica las probabilidades que en el futuro se encuentra el operador esté en cada uno de los posibles estados, ya sea ocioso, estable o inestable. Para el cálculo de esto, se utiliza la ecuación de Chapman-Kolmogórov (Papoulis, 1984) descrita en la subsección 2.5.1.

El Algoritmo 4.2 describe el cálculo de la distribución estacionaria, cuya entrada es la matriz de transición de un operador del SPS.

Antes de realizar el cálculo, era importante analizar si efectivamente existían transiciones en todos los estados, debido que existía la posibilidad que no hubiera alguna transición a algún estado en un período de tiempo. Por ejemplo, podría ser que en cierto período nunca se ha encontrado ocioso el sistema, pero si estable e inestable. Como el cálculo de la distribución estacionaria requiere un estado de inicio, se verificó si efectivamente existía o no el estado, y en caso no existir, el estado de inicio será alguno existente.

La cantidad de iteraciones que debe realizarse para el cálculo correspondiente, se proporcionó como entrada del algoritmo según lo que se estimaba necesario. Es importante destacar que entre mayor cantidad de iteraciones, mayor precisión en el valor, pero mayor es el tiempo de cómputo. Debido a esto, es que se trató de considerar un punto medio, de tal manera que tuviera bajo margen de error, pero no fueran alto su tiempo de ejecución, siendo 100.000 la cantidad de iteraciones escogida para la implementación.

---

**Algoritmo 4.2:** Cálculo de la distribución estacionaria de la cadena de Markov de un operador  $\phi$ .

---

**Entrada:**  $\Gamma$  Matriz de transición del operador  $\phi$  y  $v$  cantidad de iteraciones deseadas.

**Salida:**  $\Delta$  Distribución estacionaria de la cadena de Markov del operador  $\phi$ .

```

1:  $i \leftarrow 0$  //Estado inicial de iteración
2: for  $j = 1$  a  $3$  do
3:   if  $\Gamma_{j,x} = 0; x = 1, 2, 3$  then
4:      $i \leftarrow j$ 
5:   end if
6: end for
7:  $\tau \leftarrow \text{Arreglo}[3]$  //Contador para la normalización de los datos
8: for  $k = 0$  a  $v$  do
9:    $u = \text{randomUniform}(0, 1)$ 
10:   $\sigma = 0$ 
11:  for  $j = 0$  a  $3$  do
12:     $\sigma = \sigma + \Gamma_{i,j}$ 
13:    if  $u \leq \sigma$  then
14:       $\tau_j ++$ 
15:       $i \leftarrow j$ 
16:      break
17:    end if
18:  end for
19: end for
20:  $\Delta \leftarrow \text{Arreglo}[3]$  //Distribución estacionaria de la cadena de Markov del operador  $\phi$ 
21: for  $k = 0$  a  $3$  do
22:    $\Delta_k \leftarrow \tau_k / v$ 
23: end for
24: return  $\Delta$ 

```

---

Obtenida la distribución estacionaria, se procede a analizar las probabilidades obtenidas y como influye al operador. Para esto, se consideró que las probabilidades tuvieron entre ellas una desviación estándar superior a 0.25, debido que si esto se cumple, la probabilidad mayor de la distribución estacionaria no posee incertidumbre, porque en caso que no supere la desviación estándar, puede ser que dos probabilidades sea muy parecidas y la probabilidad no sea un comportamiento determinante. En el Algoritmo 4.3 se describe el análisis que se realiza a la distri-

bución estacionaria, siendo en primer lugar el análisis estadístico de las probabilidades, y segundo la obtención del estado con mayor valor de las probabilidades, retornando finalmente el estado del operador. Cabe destacar que el primer estado se consideró ocioso, el segundo estable, y el tercero inestable.

---

**Algoritmo 4.3:** Algoritmo predictivo del sistema de distribución de carga.

---

**Entrada:**  $\Delta$  Distribución estacionaria de la cadena de Markov del operador  $\phi$ .

**Salida:** Estado a futuro del operador, donde -1 significa estado ocioso, 0 estable y 1 inestable.

```

1: if  $\sigma(\Delta_1, \Delta_2, \Delta_3) > 0.25$  //Desviación estándar de las probabilidades de la distribución
   estacionaria then
2:    $i \leftarrow getStateMax(\Delta)$  //Obtención del estado con mayor probabilidad
3:   if  $i = 1$  then
4:     return -1
5:   else if  $i = 2$  then
6:     return 0
7:   else
8:     return 1
9:   end if
10: end if
11: return 0

```

---

## 4.5 ADMINISTRACIÓN DEL SISTEMA

El último componente del sistema es el administrador de réplicas, cuya función es administrar la cantidad de réplicas en cada uno de los operadores según los recursos disponibles por parte del sistema y según el estado que adopte un operador, ya sea a futuro o en el momento.

Para esto, se diseñó un administrador que ejecutara un tipo de algoritmo (reactivo o predictivo) según el período del ciclo que se encuentre el sistema. Cada ciclo posee 20 períodos, donde los primeros 19 corresponde al algoritmo reactivo y el último corresponde al algoritmo predictivo. Cada período posee un intervalo de 5 segundos, de esta manera, cada ciclo tendrá un intervalo de 100 segundos, la cantidad necesaria para obtener las muestras para el algoritmo reactivo, suponiendo que cada muestra es obtenida en 1 segundo.

En el Algoritmo 4.4 está el procedimiento de administración, donde primero analiza que tipo de algoritmo debe ejecutar según el período del ciclo. En

caso de realizarse el reactivo, se analiza si existen dos alertas consecutivas del mismo estado del sistema, ya sea ocioso o inestable, y de ser así, realizar una modificación a la cantidad de réplicas del operador. Por otra parte, de ejecutarse el módulo predictivo, se analiza cual fue la predicción, por lo que si es ocioso disminuirá la cantidad de réplicas y si es inestable las aumentará. Como el proceso de predicción se realiza con menor frecuencia y posee mayor cantidad de cómputo, se considero que debía crear o remover mayor cantidad de réplicas que en el módulo reactivo, aprovechando así el análisis de la historia del operador.

---

**Algoritmo 4.4:** Administración de réplicas de un operador  $\phi$  dado su comportamiento en el sistema de distribución de carga.

---

**Entrada:** Operador  $\phi$  a analizar y  $\iota$  período en que se encuentra el sistema de distribución de carga.

**Salida:** Cantidad de réplicas a modificar del operador.

```

1: if  $\iota \bmod 20 \neq 0$  then
2:    $\delta_i \leftarrow \text{AlgoritmoReactivo}(\phi)$ 
3:   if  $\delta_i$  and  $\delta_{i-1}$  son estado inestable then
4:     if No excede la cantidad máxima de réplicas en el sistema then
5:       return Crear una réplica del operador  $\phi$ 
6:     end if
7:   else if  $\delta_i$  and  $\delta_{i-1}$  son estado ocioso then
8:     return Remover una réplica del operador  $\phi$ 
9:   end if
10: else
11:    $\delta_i \leftarrow \text{AlgoritmoPredictivo}(\phi)$ 
12:   if  $\delta_i$  es estado inestable then
13:     if No excede la cantidad máxima de réplicas en el sistema then
14:       return Crear cinco réplicas del operador  $\phi$ 
15:     end if
16:   else if  $\delta_i$  es estado ocioso then
17:     return Remover cinco réplicas del operador  $\phi$ 
18:   end if
19: end if
20: return No hacer nada al operador  $\phi$ 

```

---

Dentro de las consideraciones que se tuvieron para el diseño del administrador, fue la cantidad máxima de réplicas que podían realizarse. Dado que una de las limitantes de este trabajo fue que sólo se utilizó una máquina, la cantidad de recursos son limitados, por lo que aumentar la cantidad de réplicas indefinidamente iba a generar una sobrecarga en los recursos disponibles por parte de la máquina, habiendo fallas en el funcionamiento del SPS.

## CAPÍTULO 5. EXPERIMENTOS Y EVALUACIÓN

### 5.1 IMPLEMENTACIÓN DEL SISTEMA

Para la implementanci3n del sistema propuesto, se utiliz3 como base el motor de procesamiento de *stream* S4 (S4, 2014), cuyo modelo fue explicado en la secci3n . Para el funcionamiento del sistema de distribuci3n de carga, se desarroll3 en Java, donde se incluy3 en el c3digo fuente de S4, de esta manera, fue parte del SPS, de tal manera que fuera autom3tico y transparente.

Dado que en el dise1o se propuso un balance de carga seg1n la t3cnica de r3plica, fue necesario modificar la forma en que enviaba S4 el flujo a los distintas r3plicas del operador. Por lo tanto, lo que se implemento fue un sistema que pudiera escoger la r3plica que tuviera menor tama1o en la cola de espera al momento de enviar un dato, de tal manera de escoger siempre al operador con menor carga, y en caso que posean el mismo tama1o, utilizar la primera r3plica disponible.

En la Figura 5.1 se explica gr3ficamente la distribuci3n de la carga. En la parte (a) el operador A env1a un evento al operador B, el cual posee tres r3plicas, cuyas menores colas est3n en la r3plica 2 y 3, como son iguales las colas de estas r3plicas, se escoge la primera r3plica disponible, es decir, la r3plica 2. Posteriormente, como la r3plica 2 aumento su cola, la r3plica 3 es la que posee menor cantidad de cola, como se demuestra en la parte (b), por lo tanto, es la r3plica candidata a recibir el dato enviado. Y finalmente, en la parte (c) todas las r3plicas posee el mismo tama1o de la cola, por lo tanto, se procede a enviar a la primera r3plica.

En el Algoritmo 5.1 est3 descrito la distribuci3n de carga planteada anteriormente, la cual fue implementada en S4 para realizar los experimentos seg1n lo dise1ado en el planteamiento de los algoritmos.

---

**Algoritmo 5.1:** Distribuci3n de carga entre las r3plicas de un operador.

---

**Entrada:** Evento  $\epsilon$  y operador  $\phi$ .

**Salida:** Env1o del evento a la r3plica disponible del operador  $\phi$ .

1:  $\theta \leftarrow \min \text{TamanoCola}(\phi)$  //Se escoge la r3plica que posea menor cola

2:  $\text{envioEvento}(\epsilon, \theta)$

---

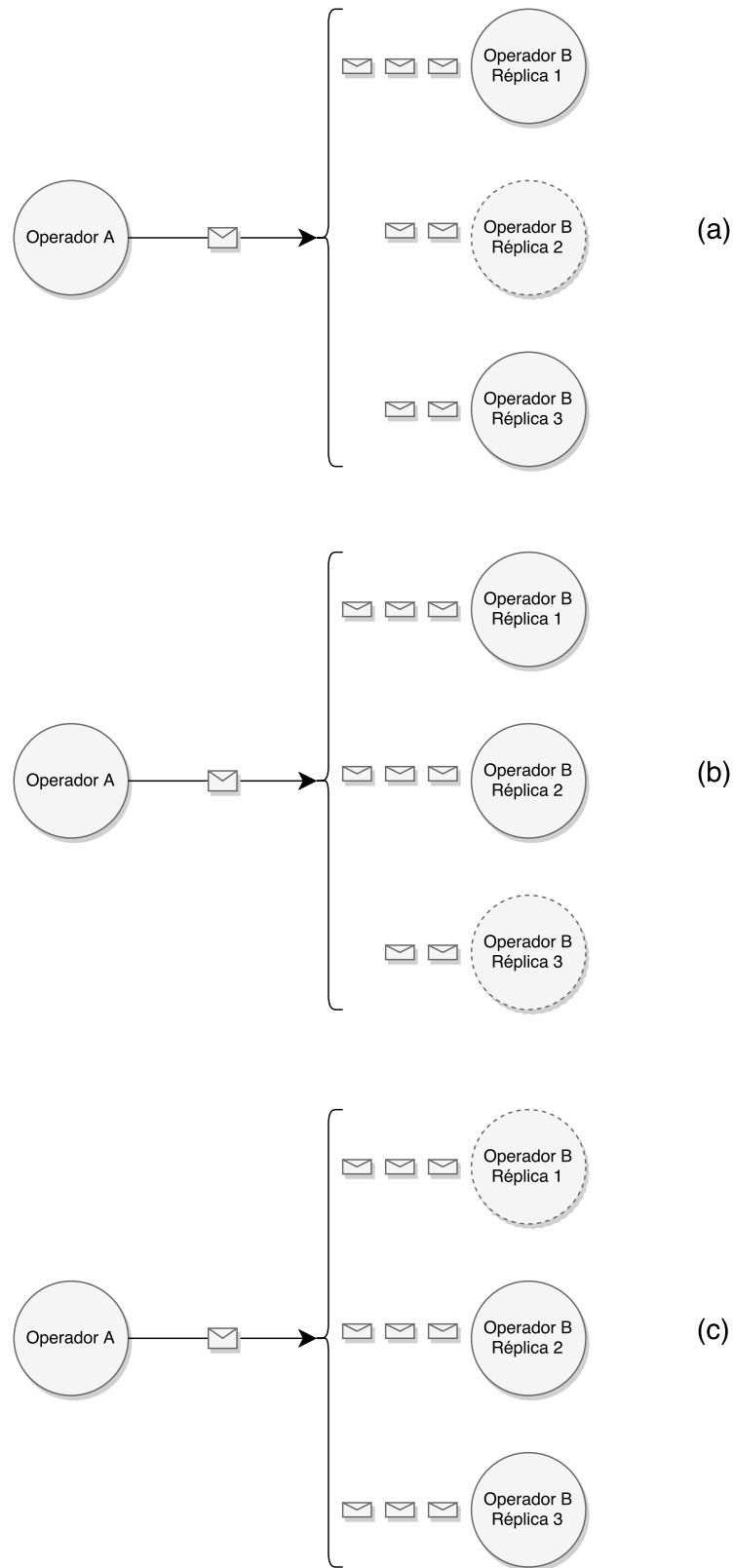


Figura 5.1: Distribución de la carga entre las réplicas.

Por otra parte, como ya se había mencionado, el sistema de distribución de carga fue parte del código fuente del proyecto de S4. Para el funcionamiento de éste, se procedió a ejecutar dos tareas: enviar las muestras para la historia del operador y enviar las estadísticas del operador, las cuales poseían un intervalo de ejecución de 1 y 5 segundos respectivamente. En el caso del envío de las estadísticas, después de ser enviadas, se consulta el estado de cada uno de los operadores, y en caso de ser necesario, se realiza las modificaciones necesarias en el sistema, ya sea de crear o eliminar réplicas de un operador. Los códigos de crear o eliminar réplicas se encuentra adjuntos en el Anexo B.

Debido al funcionamiento de los SPS, existe una fuente de datos, por lo tanto, para la correcta sincronización de las estadísticas, se procedió a realizar una espera por parte del monitor hasta que la fuente de datos esté lista para enviar los datos.

## 5.2 EXPERIMENTOS

Bla bla bla

### 5.2.1 App 1

Bla bla bla

### 5.2.2 App 2

Bla bla bla

### **5.2.3 App 3**

Bla bla bla

## **5.3 EVALUACIÓN**

Bla bla bla



## REFERENCIAS

- Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., & Zdonik, S. B. (2003). Aurora: a new model and architecture for data stream management. *VLDB J.*, 12(2), 120–139.
- Alves, D., Bizarro, P., & Marques, P. (2010). Flood: Elastic streaming mapreduce. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, (pp. 113–114).
- Andrade, H., Gedik, B., & Turaga, D. (2014). *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. Cambridge University Press.
- Appel, S., Frischbier, S., Freudenreich, T., & Buchmann, A. P. (2012). Eventlets: Components for the integration of event streams with SOA. In *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA), Taipei, Taiwan, Diciembre 17-19, 2012*, (pp. 1–9).
- Bhuvanagiri, L., Ganguly, S., Kesh, D., & Saha, C. (2006). Simpler algorithm for estimating frequency moments of data streams. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, (pp. 708–713).
- Birman, K. P. (2012). *Guide to Reliable Distributed Systems - Building High-Assurance Applications and Cloud-Hosted Services*. Texts in Computer Science. Springer.
- Bose, S. K. (2013). *An introduction to queueing systems*. Springer Science & Business Media.
- Breuer, L., & Baum, D. (2005). *An introduction to queueing theory and matrix-analytic methods*. Springer.
- Brucker, P. (2004). *Scheduling algorithms*. Springer.
- Casavant, T. L., & Kuhl, J. G. (1988). A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Software Eng.*, 14(2), 141–154.

- Chen, C. L. P., & Zhang, C. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Inf. Sci.*, 275, 314–347.
- Ching, W. K., & Ng, M. K. (2006). *Markov chains*. Springer.
- De Sapio, R. (1978). Calculus for the life sciences.
- Dong, F., & Akl, S. G. (2006). Scheduling algorithms for grid computing: State of the art and open problems.
- Dong, M., Tong, L., & Sadler, B. M. (2007). Information retrieval and processing in sensor networks: Deterministic scheduling versus random access. *IEEE Transactions on Signal Processing*, 55(12), 5806–5820.
- Falk, M., Marohn, F., Michel, R., Hofmann, D., Macke, M., Tewes, B., & Dinges, P. (2012). A first course on time series analysis: examples with sas.
- Fernandez, R. C., Migliavacca, M., Kalyvianaki, E., & Pietzuch, P. (2013). Integrating scale out and fault tolerance in stream processing using operator state management. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, (pp. 725–736).
- Ganguly, S. (2009). Deterministically estimating data stream frequencies. In *Combinatorial Optimization and Applications, Third International Conference, COCOA 2009, Huangshan, China, June 10-12, 2009. Proceedings*, (pp. 301–312).
- Gedik, B., Schneider, S., Hirzel, M., & Wu, K. (2014). Elastic scaling for data stream processing. *IEEE Trans. Parallel Distrib. Syst.*, 25(6), 1447–1463.
- Gong, Z., Gu, X., & Wilkes, J. (2010). PRESS: predictive elastic resource scaling for cloud systems. In *Proceedings of the 6th International Conference on Network and Service Management, CNSM 2010, Niagara Falls, Canada, October 25-29, 2010*, (pp. 9–16).
- Gulisano, V., Jiménez-Peris, R., Patiño-Martínez, M., Soriente, C., & Valduriez, P. (2012). Streamcloud: An elastic and scalable data streaming system. *IEEE Trans. Parallel Distrib. Syst.*, 23(12), 2351–2365.

- Gupta, D., & Bepari, P. (1999). Load sharing in distributed systems. In *In Proceedings of the National Workshop on Distributed Computing*.
- Hawwash, B., & Nasraoui, O. (2014). From tweets to stories: Using stream-dashboard to weave the twitter data stream into dynamic cluster models. In *Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine 2014, New York City, USA, Agosto 24, 2014*, (pp. 182–197).
- Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2010). Metodología de la investigación. *México: Editorial Mc Graw Hill*.
- Hirzel, M., Soulé, R., Schneider, S., Gedik, B., & Grimm, R. (2013). A catalog of stream processing optimizations. *ACM Comput. Surv.*, 46(4), 46:1–46:34.
- Ishii, A., & Suzumura, T. (2011). Elastic stream computing with clouds. In *IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA, 4-9 July, 2011*, (pp. 195–202).
- Karp, R. M., Shenker, S., & Papadimitriou, C. H. (2003). A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28, 51–55.
- Lehrig, S., Eikerling, H., & Becker, S. (2015). Scalability, elasticity, and efficiency in cloud computing: a systematic literature review of definitions and metrics. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA'15 (part of CompArch 2015), Montreal, QC, Canada, May 4-8, 2015*, (pp. 83–92).
- Leibiusky, J., Eisbruch, G., & Simonassi, D. (2012). *Getting Started with Storm - Continuous Streaming Computation with Twitter's Cluster Technology*. O'Reilly.
- Madsen, K. G. S., Thyssen, P., & Zhou, Y. (2014). Integrating fault-tolerance and elasticity in a distributed data stream processing system. In *Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014*, (p. 48).

- Neumeyer, L., Robbins, B., Nair, A., & Kesari, A. (2010). S4: distributed stream computing platform. In *ICDMW 2010, The 10th IEEE International Conference on Data Mining Workshops, Sydney, Australia, 14 December 2010*, (pp. 170–177).
- Nguyen, H., Shen, Z., Gu, X., Subbiah, S., & Wilkes, J. (2013). AGILE: elastic distributed resource scaling for infrastructure-as-a-service. In *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013*, (pp. 69–82).
- Oberhelman, D. (2007). Coming to terms with Web 2.0. *Reference Reviews*, 21, 5–6.
- Papoulis, A. (1984). *Probability, Random Variables, and Stochastic Processes*. McGraw Hill.
- Pittau, M., Alimonda, A., Carta, S., & Acquaviva, A. (2007). Impact of task migration on streaming multimedia for embedded multiprocessors: A quantitative evaluation. In *Proceedings of the 2007 5th Workshop on Embedded Systems for Real-Time Multimedia, ESTImedia 2007, October 4-5, Salzburg, Austria, conjunction with CODES+ISSS 2007*, (pp. 59–64).
- Rogaway, P., & Shrimpton, T. (2004). Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, (pp. 371–388).
- S4 (2014). Distributed stream computing platform. [Online] <http://incubator.apache.org/s4/>.
- Samza, A. (2014). Samza. [Online] <http://samza.incubator.apache.org/>.
- Schneider, S., Andrade, H., Gedik, B., Biem, A., & Wu, K. (2009). Elastic scaling of data parallel operators in stream processing. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, (pp. 1–12).

- Shahrivari, S. (2014). Beyond batch processing: Towards real-time and streaming big data. *Computing Research Repository*, *abs/1403.3375*.
- Sheu, T., & Chi, Y. (2009). Intelligent stale-frame discards for real-time video streaming over wireless ad hoc networks. *EURASIP J. Wireless Comm. and Networking*, 2009.
- Stonebraker, M., Çetintemel, U., & Zdonik, S. B. (2005). The 8 requirements of real-time stream processing. *SIGMOD Record*, 34(4), 42–47.
- Storm (2014). Distributed and fault-tolerant realtime computation. [Online] <http://storm.incubator.apache.org/>.
- Sturm, R., Morris, W., & Jander, M. (2000). Foundations of Service Level Management.
- Tanenbaum, A. S., & van Steen, M. (2007). *Distributed Systems - Principles and paradigms*. Pearson Education.
- Taylor, H. M., & Karlin, S. (2014). *An introduction to stochastic modeling*. Academic press.
- Wenzel, S. (2014). App'ification of enterprise software: A multiple-case study of big data business applications. In *Business Information Systems - 17th International Conference, BIS 2014, Larnaca, Cyprus, Mayo 22-23, 2014. Proceedings*, (pp. 61–72).
- Xing, Y., Zdonik, S. B., & Hwang, J. (2005). Dynamic load distribution in the borealis stream processor. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, (pp. 791–802).
- Xu, J., Chen, Z., Tang, J., & Su, S. (2014). T-storm: Traffic-aware online scheduling in storm. In *IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014, Madrid, Spain, June 30 - July 3, 2014*, (pp. 535–544).



## ANEXO A. CONFORMACIÓN DE MATRIZ DE TRANSICIÓN

En el Algoritmo A.1 se puede apreciar la conformación de la matriz de transición dado la historia de un operador determinado.

---

**Algoritmo A.1:** Algoritmo para la conformación de la matriz de transición.

---

**Entrada:**  $\rho$  Historial de procesamiento de tamaño  $n$  del operador  $\phi$ .

**Salida:**  $\Gamma$  Matriz de transición del operador  $\phi$ .

```

1:  $\Gamma \leftarrow \text{Matriz}[3 \times 3]$  //Matriz de transición
2:  $\tau \leftarrow \text{Arreglo}[3]$  //Contador para la normalización de los datos
3: for  $i = 1$  a  $n$  do
4:   if  $\rho_i < 0.5$  and  $\rho_{i+1} < 0.5$  then
5:      $\Gamma_{1,1}++$ 
6:      $\tau_1++$ 
7:   else if  $\rho_i < 0.5$  and  $0.5 \leq \rho_{i+1} \leq 1$  then
8:      $\Gamma_{1,2}++$ 
9:      $\tau_1++$ 
10:  else if  $\rho_i < 0.5$  and  $\rho_{i+1} > 1$  then
11:     $\Gamma_{1,3}++$ 
12:     $\tau_1++$ 
13:  else if  $0.5 \leq \rho_i \leq 1.5$  and  $\rho_{i+1} < 0.5$  then
14:     $\Gamma_{2,1}++$ 
15:     $\tau_2++$ 
16:  else if  $0.5 \leq \rho_i \leq 1.5$  and  $0.5 \leq \rho_{i+1} \leq 1.5$  then
17:     $\Gamma_{2,2}++$ 
18:     $\tau_2++$ 
19:  else if  $0.5 \leq \rho_i \leq 1.5$  and  $\rho_{i+1} > 1.5$  then
20:     $\Gamma_{2,3}++$ 
21:     $\tau_2++$ 
22:  else if  $\rho_i > 1$  and  $\rho_{i+1} < 0.5$  then
23:     $\Gamma_{3,1}++$ 
24:     $\tau_3++$ 
25:  else if  $\rho_i > 1$  and  $0.5 \leq \rho_{i+1} \leq 1.5$  then
26:     $\Gamma_{3,2}++$ 
27:     $\tau_3++$ 
28:  else
29:     $\Gamma_{3,3}++$ 
30:     $\tau_3++$ 
31:  end if
32: end for
33: for  $i = 1$  a  $3$  do
34:   if  $\tau_i \neq 0$  then
35:    for  $j = 1$  a  $3$  do
36:       $\Gamma_{i,j} \leftarrow \Gamma_{i,j} / \tau_i$ 
37:    end for
38:   end if
39: end for
40: return  $\Gamma$  //Retorno de la Matriz de transición normalizada, la cual define la cadena de Markov

```

---





## ANEXO B. MODIFICACIONES AL CÓDIGO FUENTE DE S4

En el Código B.1 se presenta la implementación de las dos tareas existentes, donde una de ellas está a cargo de obtener las muestras para el historia, y la otra está a cargo de enviar las estadísticas de los PE existentes en el sistema.

```
1 private void startMonitor() {
2     if (runMonitor) {
3         synchronized (getBlockAdapter()) {
4             try {
5                 getBlockAdapter().wait();
6             } catch (InterruptedException e) {
7                 getLogger().error(e.getMessage());
8             }
9
10            ScheduledExecutorService getEventCount = Executors
11                .newSingleThreadScheduledExecutor();
12            getEventCount.scheduleAtFixedRate(new OnTimeGetEventCount(),
13                1000, 1000, TimeUnit.MILLISECONDS);
14
15            ScheduledExecutorService sendStatus = Executors
16                .newSingleThreadScheduledExecutor();
17            sendStatus.scheduleAtFixedRate(new OnTimeSendStatus(), 6000,
18                5000, TimeUnit.MILLISECONDS);
19        }
20    } else {
21        ScheduledExecutorService sendStatus = Executors
22            .newSingleThreadScheduledExecutor();
23        sendStatus.scheduleAtFixedRate(new OnTimeSendStatus(), 6000, 5000,
24            TimeUnit.MILLISECONDS);
25    }
26
27 }
```

Código B.1: Tareas que ejecutan el sistema de distribución de carga.

En el Código B.2 se muestra la implementación que realizada para añadir una réplica a un PE en específico. El tipo *StatusPE* hace referencia un objeto creado

en la implementación, para almacenar los datos y estadísticas correspondientes al PE en el análisis de carga según el sistema de distribución de carga, como la cantidad de réplicas deseadas.

```
1 public void addReplication(StatusPE statusPE) {
2     for (Streamable<Event> stream : getStreams()) {
3         for (ProcessingElement PEPrototype : stream.getTargetPEs()) {
4             if (PEPrototype.getClass().equals(statusPE.getPE())) {
5                 for (long i = PEPrototype.getNumPEInstances(); i < statusPE
6                     .getReplication(); i++) {
7                     PEPrototype.getInstanceForKey(Long.toString(i));
8                 }
9             }
10        }
11    }
12 }
```

Código B.2: Añadir réplicas a un PE en S4.

En el Código B.3 se muestra la implementación que realizada para eliminar una réplica a un PE en específico.

```
1 public void removeReplication(StatusPE statusPE) {
2     for (Streamable<Event> stream : getStreams()) {
3         for (ProcessingElement PEPrototype : stream.getTargetPEs()) {
4             if (statusPE.getPE().equals(PEPrototype.getClass())) {
5                 for (int i = statusPE.getReplication(); i < PEPrototype
6                     .getInstances().size(); i++) {
7                     ProcessingElement peCurrent = PEPrototype
8                         .getInstanceForKey(Integer.toString(i));
9                     peCurrent.close();
10                }
11            }
12        }
13    }
14 }
```

Código B.3: Eliminar réplicas a un PE en S4.