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

Trabajo de titulación presentado en
conformidad a los requisitos para
obtener el grado de Magíster en
Ingeniería Informática

Santiago – Chile

2015

Dedicado...

AGRADECIMIENTOS

Agradezco a ...

RESUMEN

resumenBlaBlaBla

Palabras Claves: SPS;Elasticidad;Algoritmos reactivos

ABSTRACT

abstractBlaBlabla

Keywords: SPS;Elastic;Algorithm reactive;Algorithm predictive

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

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).

Entre los sistemas actuales de procesamiento de *streaming* 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 varia 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 ésto 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 motores 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 procesamiento de motores 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 motor de procesamiento de *streaming* a definir.
- Se evaluará con al menos dos aplicaciones bajo escenarios simulados utilizando datos reales.
- La distribución de flujo de datos será a nivel de operadores y no de nodos físicos, por lo que no se analizará 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 considerará el costo de comunicación de manera igualitaria para todos los operadores.
- Se comparará la solución con dos motores de procesamiento de *stream* del estado del arte.

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 para que estos sean procesados, y en caso de estar ocupado el servicio, dejar los datos en cola. Generalmente, esto se utilizado en la interacción de usuarios en la Web, como redes sociales, o reproducción *online* de material 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.

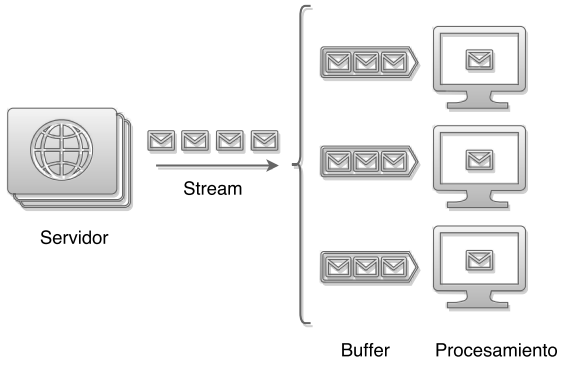

FIGURA 2.1: Flujo de datos entre servidor y clientes.

Este tipo de técnica es útil cuando se desea procesar información en tiempo real, por lo que la temporalidad de los datos es importante, como en la reproducción *online* de datos multimedia. Los datos emanados por el *streaming* pueden ser utiizados 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 en específicos, como campañas políticas o

desastres naturales.

En el procesamiento de *stream*, como monitoreo de signos vitales, detección de fraudes, reproducción de videos *online*, es necesario cumplir con ciertos 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 algo tratar de procesar los datos, sea tan grande la cantidad de datos a procesaro que no se pueden guardar en una base de datos y realizar *bash processing*. De esta forma, el utilizar *stream processing* soluciona el problema, dado que va procesando mientras 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 con datos antiguos, debido a la latencia que existe en el sistema, lo ideal es que se acerquen lo más posible 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 constantemente esté procesando los datos para el propósito dado. Por ejemplo, si se posee un sistema de análisis de partículas en el espacio, es necesario que posee una tolerancia a falla, debido que en caso que suceda alguna anomalía pueda seguir funcionando el sistema y procesar la información que se vaya entregando, porque en caso contrario, se perdería el objetivo del procesamiento.

2.2 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 *bash 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*. Por lo que el paradigma cambia a uno basado en grafos, donde los operadores corresponden a las vértices del grafo, y las aristas a los flujos de datos preprocesados 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).

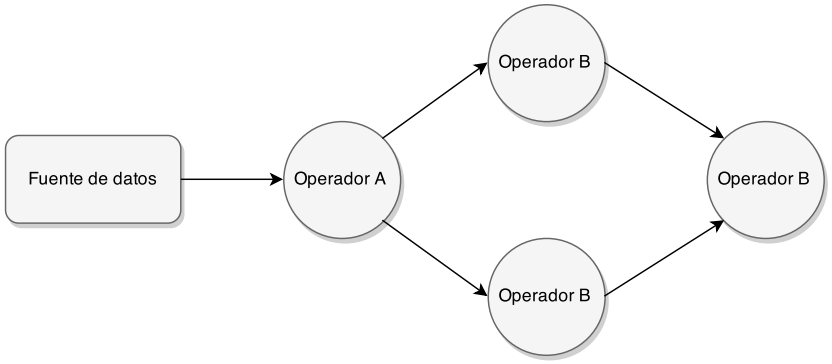

FIGURA 2.2: Ejemplo de modelo de SPS.

Cabe destacar que al ser distribuido los SPS, cada uno de las vértices del grafo serán alojadas en un nodo disponible en el ambiente que se esté almacenado el sistema, ya sea un *cluster*, un *grid* o un *Infrastructure-as-Service*. Por lo que 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. Por lo que 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 que debe ser fluida la comunicación 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 dato, 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 otro componente en el sistema.
- **Manejo de fallas en el flujo de dato** Esto significa que es importante poseer sistemas que no se preocupen de la falla 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 o pérdida de los datos.
- **Generar resultados predecibles** Cuando se realizan consultas en el sistema, puede ser que estas puedan ser correctas en cierto período de tiempo, pero debido a la falla que pueda tener el sistema, el estado de los operadores se pierde y no se obtiene el mismo resultado. Por lo tanto, es importante garantizar que el resultado sea predecible y persistente en el tiempo, de tal manera que si se

realiza una consulta, pueda volver a efectuarse un resultado igual u homólogo.

- **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 éste posee. En un contador de palabras, es importante tener variables que vayan guardando las estadísticas del sistema, por lo tanto es indispensable que posea un soporte ante esto. 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, pero estos deben ser estandarizados para el procesamiento del sistema, 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* 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** Dentro de las consideraciones que se realizan a los SPS es poder distribuir la carta de forma distribuida entre distintos procesadores o máquinas, deseando idealmente una escalabilidad incremental. 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 es importante considerar que debe atendar a respuestas lo más cercano al tiempo real, es por esto, que el procesamiento de grandes cantidades de datos debe ser de forma rápida o instantánea. Por lo que es necesario tener una optimización de la sobrecarga que pueda darse, de tal manera que no se haya un alto *overhead* en el sistema.

Cada motor 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.

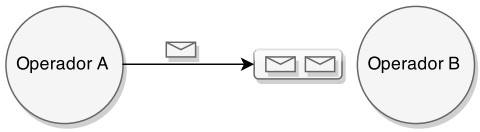

FIGURA 2.3: Modelo push.

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.

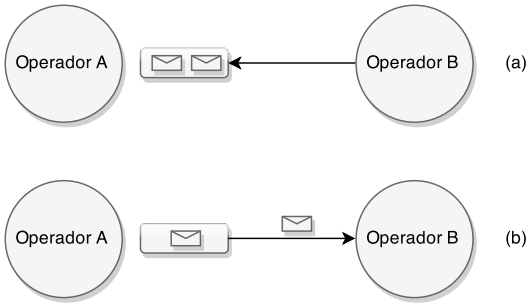

FIGURA 2.4: Modelo pull.

2.3 PROCESOS ESTOCÁSTICOS

Se define proceso estocástico como una colección de variables aleatorias X_t, con $t \epsilon T$, 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 dependan de la historia (Taylor & Karlin, 2014).

Por lo tanto, se puede definir estado como el posible comportamiento que puede tener una variable aleatoria en el sistema. Por ejemplo, se puede poseer un modelo que considere tres estados: estable, inestable y ocioso, y según el valor de la variable aleatoria pueda definirse el estado en uno de esos tres estados. Dado estos estados, puede generarse modelos que aplican estos estados como las cadenas de Markov, las cuales toman representan distintos estados dada variables aleatorias en tiempos discretos y una serie temporal con sus respectivas variables aleatorias (De Sapio, 1978).

2.3.1 Cadena de Markov

Sea X_t el valor de una variable aleatoria X en un tiempo t. El conjunto de todas 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 ω sólo depende del estado, lo cual se denota en la Ecuación 2.1 y gráficamente en la Figura 2.5. Cabe destacar que este tipo de proceso es un caso específico de los procesos estocásticos.

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

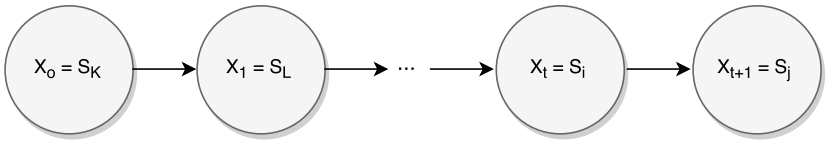

FIGURA 2.5: Proceso de Markov.

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

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

La cual se define por sus probabilidades de transición, definida en la Ecuación 2.2. En la Figura 2.6 se muestra un ejemplo de la transición del estado i al estado j, dada la probabilidad P_{ij}.

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

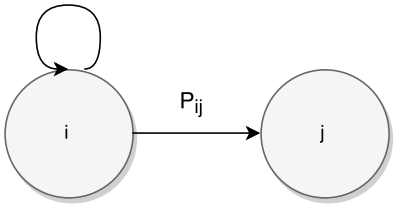

FIGURA 2.6: Cadena de Markov.

En la Ecuación 2.4 se presenta una matriz de transición de finitos estados, donde la probabilidad de pasar de un estado a otro esta terminado por una posición

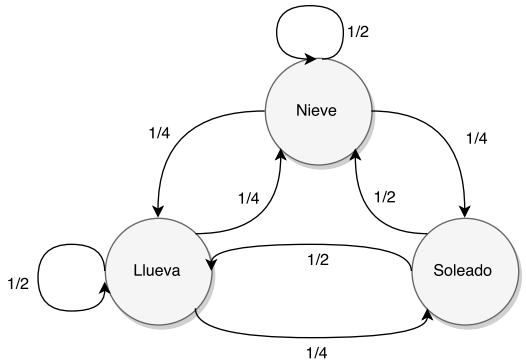

FIGURA 2.7: Ejemplo de cadena de Markov.

de la matriz, tomando en consideración que la suma de todas las transición de un estado debe ser igual a 1.

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

En la Figura 2.7 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 podrá 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.

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

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.

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

2.3.2 Trabajo relacionado

Los modelos predictivos 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 para *Cloud Computing*, lo que se analiza es 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 aplicó 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 realizó en este trabajo fue 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 analizó 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.

2.4 TEORÍA DE COLAS

Dentro de los sistemas que se basan en el modelo de productor y consumidor, puede tratarse como un modelo matemática basado en las líneas de espera en el sistema, el estudio de esto se denomina teoría de colas (Breuer & Baum, 2005).

La teoría de colas se centra en el estudio de las colas que existen en un sistema cuyo esquema existe un productor y un consumidor (servidor), como se muestra en la Figura 2.8. De esta manera, se encuentran tres entes importantes: el productor, la cola y el servidor.

- **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.

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

- **Tasa de llegada**, denotado λ, 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 μ, es la cantidad de datos, eventos o información que salen del sistema, a producto del servicio que provisto por cada servidor.
- **Tasa de rendimiento**, denotado ρ, es el porcentaje de utilización del sistema,

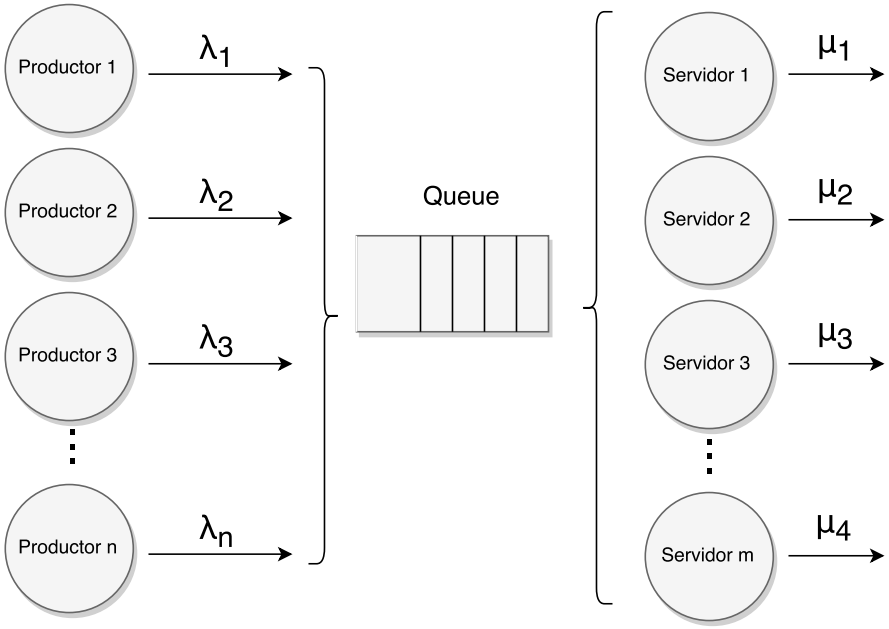

FIGURA 2.8: Ejemplo de un sistema basado en teoría de colas.

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 a causa 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 λ y una tasa de servicio μ, 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. ESTADO DEL ARTE

3.1 ELASTICIDAD

La propiedad de elasticidad en el área de *Cloud Computing* o *SPS*, está arraigado con capacidad que el sistema se adapte dinámicamente según el rendimiento que éste posea. Esto quiere decir, que aumente o disminuya los recursos que se posean para que funcione de manera eficiente el sistema.

En el caso de *Cloud Computing*, se pueden ver trabajos que han realizado 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), 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 3.1, 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 horarios de la tarde es necesario poseer una mayor cantidad de cajas disponibles que en la tarde, disminuyendo la cantidad cuando el horario borde la media noche, adaptándose de forma elástica la cantidad de cajas disponibles en el supermercado.

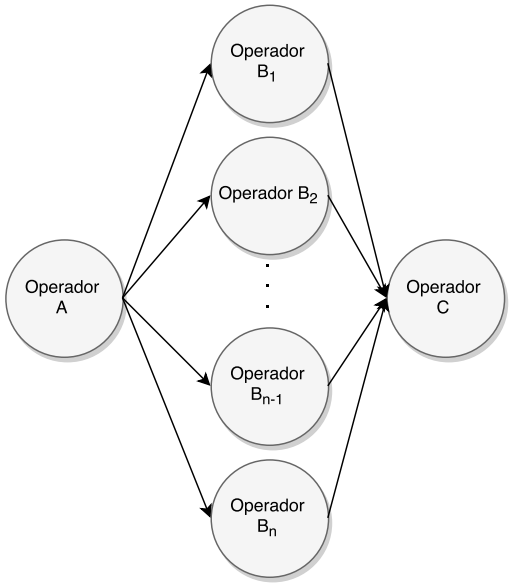

FIGURA 3.1: Elasticidad en un SPS.

3.2 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.

3.2.1 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 SLO, porcentaje de CPU utilizada o disponibilidad de la memoria (Dong & Akl, 2006).

Una de las soluciones con la perspectiva anterior es el 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. Por lo que este coordinador estará encargada de analizar los recursos del sistema, y en caso de sobrepase el umbral propuesto, se deberán emigrar 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 realizó una análisis de la cantidad de correlación que existe entre el operador y el nodo candidato, de esta manera, no necesariamente iba 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 consideró 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), el 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.

3.2.2 Recursos lógicos

A diferencia de la física, en esta perspectiva se consideran los componentes lógicos del sistema, como la carga de un operador. Por lo que las distintas soluciones que se presentan, analizan la 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 sistema. En cambio, el segundo enfoque está basado en un modelo que analizará el sistema según su estado en el transcurso de su ejecución.

3.2.3 Enfoque estático

Este enfoque se ha implementando en distintos motores de *streaming*, 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 son *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 operado. 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.

3.2.4 Enfoque dinámico

Este enfoque está basado en el estado del sistema, siendo esto el parámetro para optimizar el rendimiento del sistema (Casavant & Kuhl, 1988). Esto significa que si el sistema le surge una anomalía, como una sobrecarga o latencia entre nodos, es necesario realizar un cambio en el sistema, con el fin de solucionar estos problemas.

Por lo que para solucionar estos problemas, se consideran dos modelos: el reactivo y el 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, una 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.

3.3 TÉCNICAS DE BALANCE DE CARGA

Existen distintas técnicas de balance de carga que utilizan alguno de los dos modelos presentados anteriormente, los cuales están enfocadas en mejorar al rendimiento del sistema en caso de existir una sobrecarga. Dentro de las técnicas existentes se encuentrán la planificación determinista (Xu et al., 2014; Dong et al., 2007), descarte de eventos (Sheu & Chi, 2009), migración (Xing et al., 2005), fisión

(Gulisano et al., 2012; Ishii & Suzumura, 2011; Gedik et al., 2014; Fernandez et al., 2013) y *load balancing*.

3.3.1 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 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).

En el caso de *Streaming*, se han realizado varias análisis respecto al 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 posee (Bhuvanagiri et al., 2006).

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 generará 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 es estacionaria.

3.3.2 Load Shedding

En los SPS también se encuentra la técnica de *load shedding* o descarte de eventos, que consiste en descartar eventos del sistema en caso de existir un comportamiento anómalo, ya sea un máximo en la cantidad de la cola u otro factor como se puede apreciar en la Figura 3.2. Por ejemplo, en la transmisión de *video streaming*, al enviar el flujo de información existe un administrador que estará analizando el contenido a procesar, por lo que en caos 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).

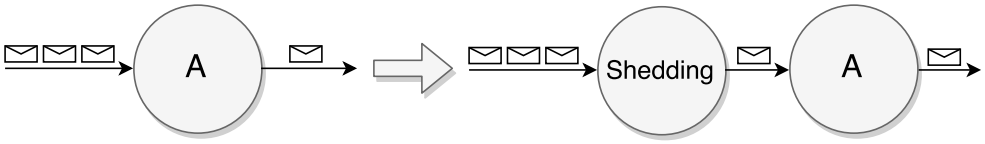

FIGURA 3.2: Load shedding en un SPS.

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 descartará 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.

3.3.3 Migración

La técnica de migración está basada en el traspaso de un operador de un nodo a otro, según el estado del sistema, como se puede apreciar en la Figura 3.3. 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 de 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).

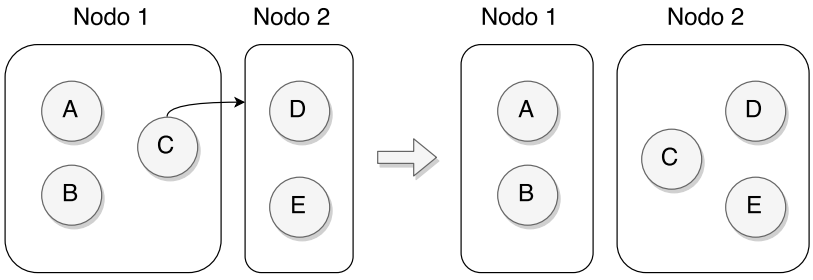

FIGURA 3.3: Técnia de migración en un SPS.

3.3.4 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 existe una sobrecarga en el operador. En Figura 3.4 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.

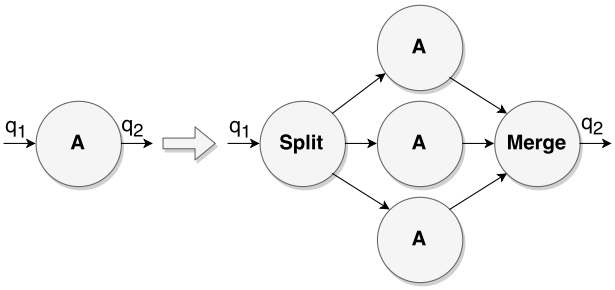

FIGURA 3.4: Técnica de fisión en un SPS.

Una aplicación que aplica la técnica de fusión según el enfoque estático, es la paralelización de tareas de Storm (Documentation, 2014), donde se indica la cantidad de operadores necesarios para realizar una tarea paralelamente. De esta manera, existe un proceso que está encargada de la tarea deseada, y n hebras por la cantidad de operadores que se deseen.

Otro sistema que abarca esta técnica, utilizando 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 cumplan 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 orden. 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. Pero no sólo este tipo de sistemas aplica este método, también se puede ver otros trabajos como (Gedik et al., 2014; Schneider et al., 2009) que 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 sobrepaso el umbral propuesto, que era en este caso la utilización de la CPU, se procede a replicar el operador sobrecargado. En la Figura 3.5 se puede ver el caso de un simple sistema, en el cual el operador u está enviado 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 y ya no es necesario más réplicas, 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.

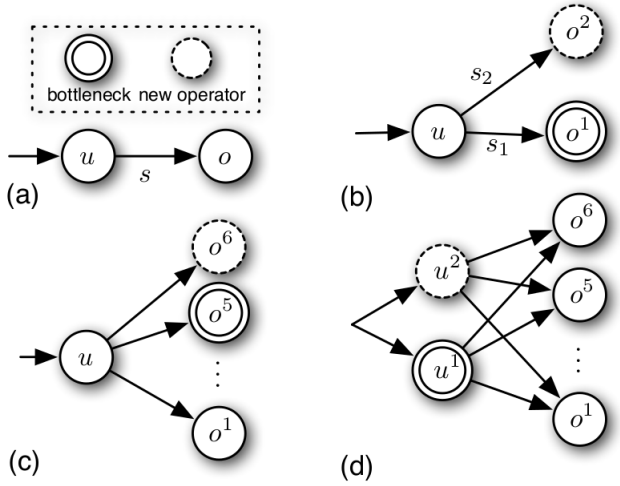

FIGURA 3.5: Ejemplo de replicación de los operadores (Fernandez et al., 2013).

CAPÍTULO 4. ESTRUCTURA DEL SISTEMA DE DISTRIBUCIÓN DE CARGA

4.1 COMPONENTES DEL SISTEMA

Los componentes que se establecen en el sistema de distribución de carga son cuatro: monitor de carga, analizador de carga, predictor de carga y administrador de réplicas, que se pueden apreciar en la Figura 4.1.

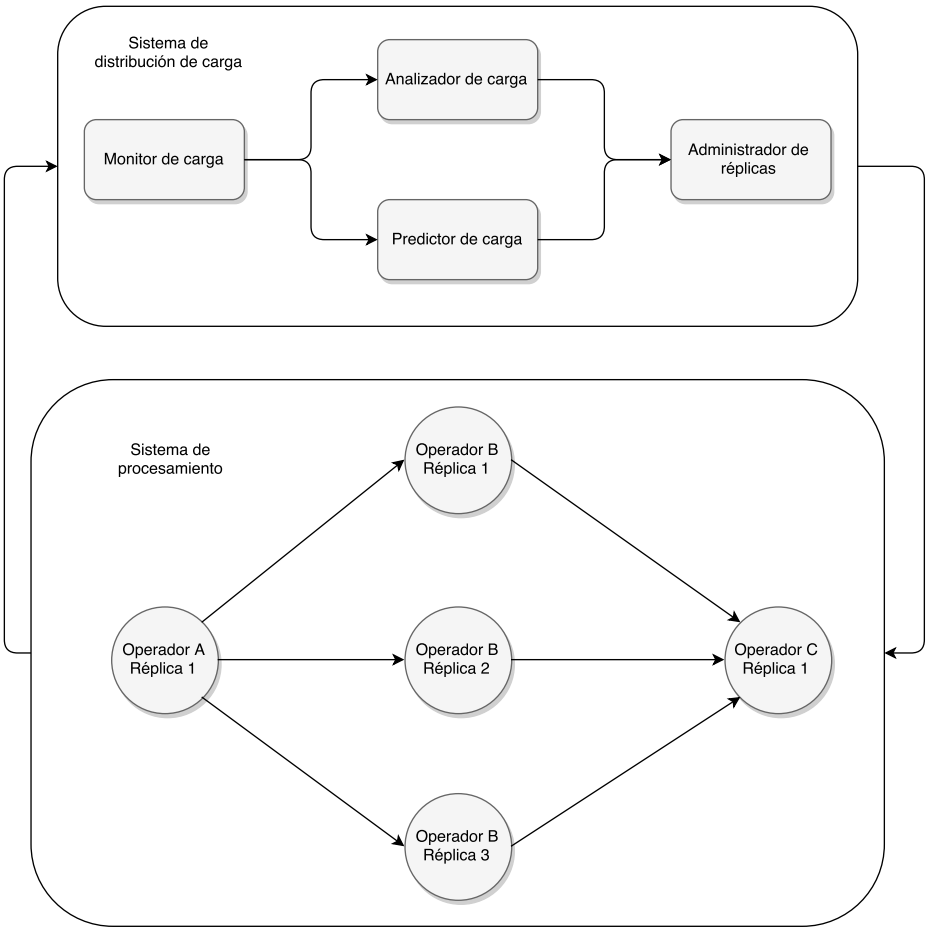

FIGURA 4.1: Estructura del sistema de distribución de carga

4.1.1 Inicialización del sistema

Antes de procesar el flujo de los distintos operadores, es necesario registrar la topología del sistema, por lo tanto cada uno de los operadores son identificados en el monitor de carga, como también la direccionamiento entre los distintos operadores.

4.1.2 Monitor de carga

Este componente está a cargo de recopilar las estadísticas de cada uno de los operadores, como la tasa de llegada, de procesamiento y eventos procesados. Para guardar las estadísticas, se utilizó un arreglo de un objeto, el cual está especificado en la Tabla 4.1.

Tipo	Atributo	Definición
long	recibeEvent	Blablabla
long	sendEvent	Blablabla
double	sendEventUnit	Blablabla
double	sendEventPeriod	Blablabla
double	processEvent	Blablabla
long	queueEvent	Blablabla
Queue¡Double¿	history	Blablabla
Class	pe	Blablabla
int	replication	Blablabla
Queue¡Integer¿	markMap	Blablabla
long	eventCount	Blablabla

TABLA 4.1: Estructura del objeto del estado de un PE.

CAPÍTULO 5. ALGORITMOS DE DISTRIBUCIÓN DE CARGA

Bla bla bla

5.1 ALGORITMO REACTIVO

bla

5.2 ALGORITMO PREDICTIVO

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).

Breuer, L., & Baum, D. (2005). *An introduction to queueing theory and matrix-analytic methods*. Springer.

Brucker, P. (2004). *Scheduling algorithms (4. ed.)*. 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.

Documentation, S. (2014). Understanding the parallelism of a storm topology. `http://storm.incubator.apache.org/documentation/Understanding-the-parallelism-of-a-Storm-topology.html`.

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*.

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).

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. URL `http://www.oreilly.de/catalog/9781449324018/index.html`

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. `http://incubator.apache.org/s4/`.

Samza, A. (2014). Samza. `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. `http://storm.incubator.apache.org/`.

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).

APÉNDICE A. MANUAL DE USUARIO

A.1 REQUERIMIENTOS

blablablabla....

A.2 INSTALACIÓN

blablablabla....

blablablabla....