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 motores 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

Entre los diferentes motores de procesamiento de datos masivos, existen los motores de *streaming*, 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 cuales luego 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 a través del cual fluye un *stream* de datos que es procesamiento por el conjunto de operadores que lo componen, los operadores corresponden a los nodos del grafo y las aristas a los flujos de datos preprocesados que salen del operador (Shahrivari, 2014).

El modelo de procesamiento que se muestra en la Figura 1.1, corresponde a un SPE (Sistema de Procesamiento de Eventos). 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).

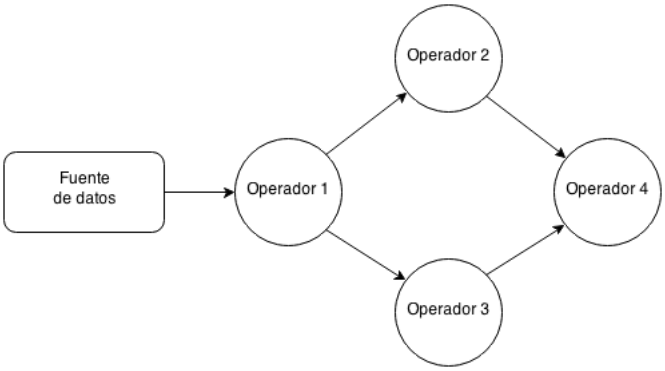

FIGURA 1.1: Ejemplo de modelo de SPE.

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 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 programa a que operador debe ir, en cambio en el segundo se debe indicar quién lo envía y quién lo recibe.

Pues bien, independiente del método que se utilice, existe un problema en la distribución, dado el dinamismo de los datos a procesar, pudiendo generarse sobrecargas en algún operador. Esto se produce dado que la tasa de procesamiento es menor a la tasa de llegada, generando colas en el sistema (Breuer & Baum, 2005). Por ejemplo, si se posee una tasa de llegada λ y una tasa de servicio μ, donde $\mu < \lambda$, se generan colas en el sistema, 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.

El caso más simple de modelo de SPE, existe una réplica por cada operador, de esta manera, cada servicio tendrá un valor de 1. Por lo tanto, al crear réplicas de cada uno de los servicios, se aumenta el rendimiento del sistema, en caso de haber colas. Para no utilizar recursos de forma excesiva, es necesario realizar el proceso de forma elástica, la cual se define como el aumento o disminución de la cantidad de servicios según la cantidad requerida por el sistema.

Es importante dar una solución al problema de sobrecarga de los operadores, para disminuir el tiempo del procesamiento y aumentar la precisión de los resultados del sistema. Por lo tanto, dada la falta de flexibilidad del modelo de procesamiento de *streams*, surge un problema basado en la sobrecargada de los operadores más demandados. Esto debido a que no existe una forma de disminuir la sobrecarga y reducir las colas de espera, para 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. En la Figura 1.2 se muestran los cuatro módulos que componen la estructura del módelo propuesto 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

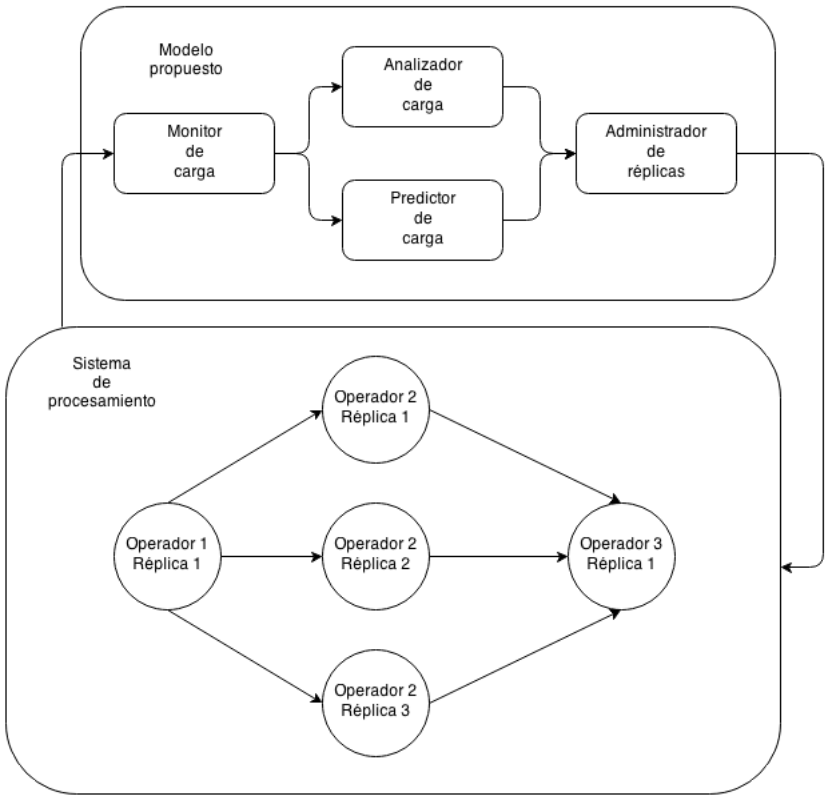

FIGURA 1.2: Estructura del sistema de operadores y modelo propuesto.

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.

Este sistema dará un procesamiento más rápido de la información entregada en el sistema de procesamiento *online* de datos, a través de un proceso de optimización que posea 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 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 SPE. 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 SPE, vale decir, los algoritmos de predicción y distribución. Cada ejecución de los experimentos se basan según los principios de un SPE.

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 8. 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 4.3.2, y para el uso de un modelo matemático se utilizará MATLAB 2014a. De forma complementaria, se utilizará Texmaker 4.2 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. ESTADO DEL ARTE

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

2.1 PERSPECTIVA FÍSICA

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 (Madsen et al., 2014). Por ejemplo, que exista una violación en el SLO, la cantidad de CPU deseada o disponibilidad de memoria, tomando estos parámetros como umbrales para determinar cuando existe una sobrecarga, para posteriormente utilizar alguna técnica y aliviar el sistema.

Una de las soluciones con la perspectiva anterior es el Borealis, donde considerará 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 con menor cantidad de carga (Xing et al., 2005). Dentro de las problemáticas que existe en este sistema era la cantidad de datos que se necesitan transferir de un nodo a otro, por lo que ellos para realizar las pruebas consideraron un caso ideal, donde no poseían limitaciones de la red.

Por lo tanto, para realizar optimizaciones al sistema se han presentando dos tipos de enfoques: el estático y el dinámico Dong & Akl (2006). El primer enfoque se centra en la confección de un modelo definido y fijo antes de iniciar el sistema y que no varía en el tiempo. En cambio, el segundo se basa en la adaptación del

sistema según su estado en tiempo de ejecución.

2.2 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 en las variables (Casavant & Kuhl, 1988). Esto significa, que al momento de distribuir los distintos operadores, esto se realiza homogéneamente, independiente de la carga, latencia u otro estado del sistema.

Storm utiliza la técnica de distribución de operadores según alguna política, tomando el enfoque estático (Storm, 2014). El sistema configura el número de operadores que son necesarios para realizar una tarea, para que después estos sean repartidos en los distintos nodos disponibles según la política de *Shuffle grouping*. Esta técnica basada en algoritmos de planificación (Brucker, 2004), consiste en distribuir los operadores en los distintos nodos utilizando una planificación *Round-Robin*, de tal manera que la cantidad de operadores sea uniforme en los nodos del sistema (Leibiusky et al., 2012).

Otra técnica es el uso de la función *hash* (Rogaway & Shrimpton, 2004) para distribuir los operadores en el grafo, como lo planteado en S4 (Neumeyer et al., 2010). Esto consiste en aplicar lo anterior a algún atributo del evento, mapeando el evento al operador que corresponda de los n operadores disponibles según el valor de la función. Cabe destacar que si no existe un operador mapeado con la imagen de la función, el sistema clona uno existente evaluando el nuevo con la imagen de la función como identificador. De esta manera, esta técnica provee dinamicidad respecto a la cantidad de operadores en el sistema.

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 alguno de ellos tenga una mayor tasa de procesamiento que la tasa de llegada. Si bien, no es una solución óptima, es un buen complemento para un modelo con el enfoque dinámico.

2.3 ENFOQUE DINÁMICO

Este enfoque está basado en el estado del sistema, donde según las variables y estado de cada uno de sus atributos, genera una acción en el sistema (Casavant & Kuhl, 1988). Esto significa que si el sistema posee alguna anomalía, como una sobrecarga en un operador o latencia entre distintos nodos, se realiza un cambio en el sistema, con el fin de solucionar estos problemas. Para poder dar una solución al problema de sobrecarga, se pueden utilizar dos tipos de modelos: reactivo y predictivo.

2.3.1 Reactivo

Este modelo está basado en la detección de sobrecargas en el sistema a través de un monitor (Gulisano et al., 2012). El monitor recibe periódicamente la carga de cada uno de los nodos u operadores, y en caso que se 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 nodo u operador (Bhuvanagiri et al., 2006). Por ejemplo, para realizar una paralelización de un operador sobrecargado, se utiliza un algoritmo que analiza si los operadores sobrepasan un umbral propuesta, el cual está determinado por la frecuencia máxima de llegada de ellos según el tasa de procesamiento del operador (Schneider et al., 2009).

2.3.2 Predictivo

El modelo predictivo está basado en modelos matemáticos que puedan simular la actividad del operador, y predecir la carga de éste. Primero se determina la carga de un operador en cierto período de tiempo, y después se aplica un modelo matemático que predice la carga en el próximo intervalo de tiempo. La predicción de carga de cada operador se realiza mediante modelos de Markov (Gong et al., 2010), y analizando si existe alguna sobrecarga en los operadores a futuro. Si bien lo anterior no ha sido implementando en el contexto de *Stream*, si se ha realizado en *Cloud Computing* según la sobrecarga de los computadores (Nguyen et al., 2013).

Existen distintas técnicas que son utilizadas en ambos modelos, entre las cuales están la planificación determinista (Xu et al., 2014; Dong et al., 2007), descarte de eventos (Sheu & Chi, 2009), migración (Xing et al., 2005), paralelización (Gulisano et al., 2012; Ishii & Suzumura, 2011; Gedik et al., 2014) y replicación (Fernandez et al., 2013).

La planificación determinista (Dong et al., 2007) se centra en la planificación según los recursos y estados del sistema *a priori* según alguna métrica (Xu et al., 2014). Una métrica utilizada es la frecuencia de datos estimada en un nodo u operador (Ganguly, 2009). Esta técnica se utiliza, por ejemplo, en *StreamIt* (Thies et al., 2002). Una de las limitaciones es que si bien realiza una predicción determinista de la frecuencia, no necesariamente es correcta a futuro, por lo que no se puede predecir las tasas de tráfico en el transcurso del procesamiento, sino sólo estimarlas al comienzo de la ejecución del sistema.

Otra estrategia está orientada a descartar eventos de un operador sobrecargado, de tal manera de no generar colas en el sistema. Esta estrategia que si bien no está implementada en el sistema por defecto de S4, puede habilitarse (S4, 2014). Otro ejemplo, es la trasmisión de vídeo *streaming*, donde se descartan los datos que son de baja calidad, para procesar en su mayoría información de alta calidad

(Sheu & Chi, 2009). Esta solución está pensada para disminuir la carga, perdiendo la exactitud de la información debido a la pérdida de datos. Por lo tanto existe una menor fiabilidad en el sistema en caso de realizar operaciones de transacción (Birman, 2012).

También se encuentra la migración, en el cual según el estado del sistema se migran los operadores de un nodo a otro. En (Xing et al., 2005) se implementa esta técnica, y si bien genera una menor carga en distintos nodos, produce un alto costo en la transferencia de los datos. Al realizar la transferencia de los datos, existe una menor tolerancia a fallos, a raíz de lo cual, se propone el uso de un búffer en el sistema, aumentando sus costos (Pittau et al., 2007).

Desde otra perspectiva, existen las técnicas de paralelización y replicación, las cuales se utilizan en caso de sobrepasar un umbral, el cual depende de la carga de un operador, nodo, entre otras variables. El primero consiste en paralelizar una tarea, la cual está determinada por un conjunto de operadores, en otro nodo físico (Ishii & Suzumura, 2011). En cambio, la replicación consiste en replicar un operador a nivel lógico del grafo (Madsen et al., 2014). Una de las características que existen en este tipo de soluciones es la elasticidad, que consiste en la capacidad de aumentar o disminuir la cantidad de operadores según la necesidad del sistema.

Una aplicación de la técnica de paralelización según el enfoque estático, es la paralelización de tareas de Storm (Documentation, 2014), donde un conjunto de operadores realizan una tarea, indicado la cantidad de tareas que se desean ejecutar paralelamente en el sistema. Un ejemplo aplicado de esta técnica según el enfoque diñamico es StreamCloud (Gulisano et al., 2012), que dada la cantidad de consultas que van llegando al sistema, se paralelizan las tareas existentes. Uno de los problemas que surge en estos casos son las operaciones con estado, como lo son los contadores o algoritmos de orden. La solución planteada es poseer un operador que realiza la tarea de *merge*, que consiste en recibir las salidas de las tareas paralelas, agrupando los datos y proporcionando una salida según lo realizado por cada uno de las operaciones (Gedik et al., 2014).

Por otra parte, se usa la técnica de replicación en una aplicación diseñada por Fernández (Fernandez et al., 2013). Ésta está enfocada en la replicación de operadores, la cual se activa si se detecta un cuello botella en el procesamiento de los datos. Para la detección, existe un monitor que está recibiendo la carga de cada uno de los procesos, y si es sobrepasado el umbral, debe realizar una petición de replicación al operador sobrecargado.

REFERENCIAS

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.

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.

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.

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.

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).

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.

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*.

Storm (2014). Distributed and fault-tolerant realtime computation. `http://storm.incubator.apache.org/`.

Thies, W., Karczmarek, M., & Amarasinghe, S. P. (2002). Streamit: A language for streaming applications. In *Compiler Construction, 11th International Conference, CC 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, (pp. 179–196).

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....