Universidad de las Ciencias Informáticas

Facultad 4

Adaptación del algoritmo Teoría de Respuesta al Ítem para la estimación del conocimiento latente sobre Datos Educacionales Masivos

Trabajo final presentado en opción por el título de

Máster en Informática Avanzada

Autor:

Ing. Orlando Grabiel Toledano López

Tutor (es):

Dr C. Rafael Arturo Trujillo Rasúa

MSc. Ángel Alberto Vázquez Sánchez

La Habana, septiembre 2018

Dedicatoria

A mi madre, mi padre y mi querido hermano.

A mi novia y futura esposa.

A mis inseparables amigos.

A mis antiguos compañeros de universidad y mis compañeros de trabajo.

A mi país por la gran educación que me ha dado durante toda mi vida.

Agradecimientos

A mi madre Ana María por su constante entrega a mi educación y crianza, por tener un amor infinito hacia mí; por ser mi luz y guía.

A mi padre Carlos Guillermo por su sabiduría, amor y comprensión, por siempre estar a mi lado en todo momento y por haberme convertido en el hombre que soy hoy.

A mi hermano Ismael por ser el hermano mayor que me acompañó en todo momento y me inculcó a andar en este apasionante mundo de la informática. Gracias por todo tu cariño y ser mi fuente de inspiración.

A mi querida novia Selianne, por convertirte en la mujer que le da significado a mi vida, por amarme y cuidarme a pesar de mis defectos.

A mis tutores Rafael Arturo y Ángel Alberto por su invaluable ayuda, principalmente en la concepción y orientación de esta investigación.

A mi otro hermano Hugo, por tener la misma pasión por las Ciencias Informáticas y estar al tanto de toda mi investigación y ser un amigo sencillamente genial e inseparable.

A mis inseparables amigos Elias, Osmar y Yasiel que me acompañan y me brindan su amistad y compañía en todo momento.

A mis amigos de infancia y la vieja escuela Rodrigo, Adrián y Julito que los considero como mis hermanos de toda la vida.

Declaración jurada de autoría

Declaro por este medio que yo Orlando Grabiel Toledano López, con carné de identidad 91101842342, soy el autor principal del trabajo final de maestría “Adaptación del algoritmo Teoría de Respuesta al Ítem para la estimación del conocimiento latente sobre Datos Educacionales Masivos”, desarrollada como parte de la Maestría en Informática Avanzada y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Y para que así conste, firmo la presente declaración jurada de autoría en La Habana a los ___ días del mes de _____ del año 2018.

_____

Ing. Orlando Grabiel Toledano López

Resumen

La constante interacción tecnológica de las personas, los dispositivos móviles y estaciones de trabajo conectadas a la red genera un gran cúmulo datos, de los cuales se puede extraer conocimiento. El sector educacional no está ajeno a este fenómeno y se puede observar cómo el uso creciente de las Tecnologías de la Información y las Comunicaciones influyen en sus procesos sustanciales. El área de la Minería de Datos Educativos permite aplicar métodos computarizados para la realización de tareas predictivas que posibilitan detectar patrones sobre datos educativos y clasificar grupos de individuos a partir de la medición de su conocimiento latente. Entre los métodos predictivos, existe Teoría de Respuesta al Ítem, el cual es un modelo matemático de medición que permite conocer, a partir de respuestas observadas de individuos sobre ítems evaluativos, la habilidad o conocimiento latente. Este modelo no es viable aplicarlo en su estado actual sobre cantidades masivas de datos, debido a que su costo computacional depende de la cantidad de estudiantes y la cantidad de ítems. Para dar respuesta a este problema se realiza una adaptación de dicho modelo mediante el uso de técnicas de procesamiento en paralelo usando la herramienta Apache Spark. Se valida la propuesta de solución mediante pruebas de rendimiento calculándose a su vez las métricas Speedup y eficiencia. Finalmente como aporte práctico de la solución se obtiene un algoritmo de estimación de conocimiento disponible a través de una API REST de servicios, con un mejor rendimiento que el algoritmo original.

Palabras clave: Algoritmo, estimación de conocimiento latente, Minería de Datos Educativos, Teoría de Respuesta al Ítem.

Abstract

The constant technological interaction of people, mobile devices and work stations connected to the network generates a large amount of data, from which knowledge can be extracted. The educational sector is not exempts to this phenomenon and it can be see how the growing use of Information and Communication Technologies influence their substantial processes. The area of Educational Data Mining allows applying computerized methods to perform predictive tasks that make it possible to detect patterns on educational data and classify groups of individuals based on the measurement of their latent knowledge. Among the predictive methods, there is Item Response Theory, which is a mathematical measurement model that allows knowing, from the observed responses of individuals on evaluative items, the skill or latent knowledge. This model is not viable to apply in its current state on massive amounts of data, because its computational cost depends on the number of students and the number of items. In order to respond to this problem, an adaptation of said model is made through the use of parallel processing techniques using the Apache Spark tool. The solution proposal is validated through performance tests, calculating in turn the Speedup and efficiency metrics. Finally, as a practical contribution of the solution, is obtained an algorithm for estimating knowledge available through a service REST API, with a better performance than the original algorithm.

Keywords: Algorithm, latent knowledge estimation, Educational Data Mining, Item Response Theory.

ÍNDICE

INTRODUCCIÓN

En los últimos años el uso creciente y masivo de las Tecnologías de la Información y las Comunicaciones (TIC) ha provocado gran impacto en la sociedad. Con el desarrollo de los dispositivos móviles, las tecnologías de conexión inalámbrica y el auge de la computación en la nube, es muy común hoy en día que las personas interactúen constantemente con el Internet y/o Red Digital de Servicios Integrados (RDSI) (Gupta, 2015), sin importar el lugar o momento. Toda esta interacción tecnológica genera huellas digitales que van acumulándose hasta originar cantidades enormes de datos (Frydenberg, 2015).

Para lograr adquirir y analizar tanta información surge el área del conocimiento conocida como Big Data, la cual se caracteriza por conjuntos de datos cuyo tamaño va más allá de la capacidad de captura, almacenado, gestión y análisis de las herramientas de base de datos. Big Data puede ser utilizado para la mejora de procesos permitiendo su simplificación y control, propiciando la reducción de costes y esfuerzos (Camargo-Vega, Camargo-Ortega, & Joyanes-Aguilar, 2015). Además, para identificar patrones de comportamiento complejos que permitan detectar ataques informáticos en tiempo real. Enfocado al soporte a la toma de decisiones, puede ser también una potente herramienta para descubrir información que antes podría estar oculta, a través de la aplicación algoritmos automáticos de minería de datos. Todas estas ventajas se pueden agrupar en una principal de las que derivan el resto: *"obtener más información/conocimiento"* a partir del análisis profundo de los datos (Manyika, Chui, & Brown, 2011).

La Educación, como sector impulsor y enriquecedor de la sociedad, no ha quedado exenta del impacto del desarrollo tecnológico, quedando expuesta, además, a las potencialidades del Big Data y su uso enfocado a fortalecer el proceso de enseñanza y aprendizaje. Este fenómeno se puede observar en dicho sector debido al uso creciente de las TIC en sus procesos vitales, donde la interacción con herramientas educativas en línea, sistemas adaptativos, Cursos Online Masivos y Abiertos (MOOCs, siglas en inglés), entre otras plataformas, genera un gran cúmulo de datos educacionales que pueden ser aprovechados para encontrar conocimiento oculto y mejorar de forma sustancial el proceso educativo docente (Dos Santos, 2015).

En el marco del contexto del análisis de datos masivos surge la Minería de Datos Educacionales (MDE), la cual se ocupa del desarrollo, investigación y aplicación de métodos computarizados para detectar patrones en grandes colecciones de datos que provienen del contexto educacional, que de otra manera podría ser difícil o imposible de analizar debido al enorme volumen en los cuales estos existen. Estos datos provienen de varias fuentes, incluidos los datos de los entornos presenciales tradicionales, el uso de software educativo, los cursos en línea y las actividades evaluativas (Cristóbal Romero, Ventura, Pechenizkiy, Baker, & S.J.d., 2011).

Con el uso de la MDE se pueden aplicar diferentes métodos para descubrir conocimiento sobre grandes fuentes de datos, tales como: Métodos predictivos, descubrimiento de estructuras y modelos, minería de relaciones y destilación de datos para juicio humano (Galindo & García, 2010). Los datos educacionales masivos comprenden datos de identidad, datos de interacción de usuarios del sistema, datos de todo el sistema y datos inferidos por los estudiantes. A partir de todos ellos se puede extraer conocimiento de un gran valor (Cristobal Romero & Ventura, 2012).

Entre los métodos predictivos existentes, resulta de gran utilidad para el sector educacional la estimación de conocimiento latente, el cual permite determinar en qué medida un estudiante domina un conocimiento o habilidad determinada en un momento dado (J. A. Larusson, 2014). Para realizar una inferencia del conocimiento que posee un estudiante, entiéndase por conocimiento lo que un estudiante conoce (o domina) que puede ser significativo para la situación de aprendizaje actual (Habilidades, Hechos, Conceptos, Principios y Esquemas) existen diferentes algoritmos, tales como: Performance Factors Analysis (PFA) (Goldin, 2015), Bayesian Knowledge Tracing (BKT) (Sande, 2013), Item Response Theory (IRT) (Yue & Hambleton, 2017). Estos proveen modelos donde a partir de datos dicotómicos[1] obtenidos por diferentes test aplicados a los estudiantes, permiten estimar conocimiento latente. Con ellos de forma general, se puede medir qué conoce un individuo en un momento específico, sus componentes de conocimiento y estimar su probabilidad de éxito frente a una pregunta o ítem en un examen (R. S. J. D. Baker, 2010).

De los algoritmos anteriormente planteados, particularmente IRT se puede aplicar en la elaboración de instrumentos evaluativos dinámicos y adaptables a las habilidades individuales de los estudiantes. Se debe tener en cuenta, que este parte de la rama de la psicología para el diseño de test mentales que permiten la medición del conocimiento humano (Pérez Gil, 2011). Desde sus inicios, ha sido una buena alternativa al modelo Teoría Clásica de Test (TCT) que era utilizado con el mismo propósito. Este supone una mejora con respecto a TCT, ya que ofrece métodos que permiten la construcción de test más adecuados, eficientes y permite evaluar en una misma escala a los ítems evaluativos y los examinados (F. B. Baker, 2001). Lo más importante de IRT es que asegura que las mediciones de conocimiento latente realizadas a las personas sean invariantes con respecto a los ítems aplicados en la medición (Güler, Uyanık, & Teker, 2014).

El algoritmo IRT utiliza tres modelos logísticos fundamentales propuestos por Rasch, Birnbaum y Fred Lord; los cuales permiten estimar la probabilidad de que un sujeto conteste correctamente un determinado ítem, teniendo en cuenta el nivel de habilidad del individuo y los parámetros del ítem.

[1] **Datos dicotómicos:** Registro de vectores de respuestas con valores binarios (ceros y uno). Se usa el valor cero si ha contestado incorrectamente y uno si ha contestado correctamente.

Un ítem puede poseer hasta tres parámetros y según la cantidad de estos se selecciona el modelo logístico adecuado (F. B. Baker, 2001).

El Grupo de Investigación de Tecnologías Aplicadas a la Educación (GITAE), perteneciente a la Facultad 4 de la Universidad de las Ciencias Informáticas, posee varias líneas de investigación, entre las que se encuentra la línea de Minería de Datos Educativos Masivos (MDEM), la cual trabaja en el desarrollo de una plataforma para aplicar MDE. La misma comprenderá una capa de administración que incluirá herramientas como: HUE para la interfaz web, Zookeper y Oozie para la coordinación y planificación del flujo de trabajo. Se incluye además una capa para el procesamiento de datos, la cual comprende herramientas para la minería de datos: Mahout, Docker y Spark Mlib. Esta capa incluirá además motores de procesamiento, entre los cuales se puede aplicar Hadoop/MapReduce o Apache Spark.

El objetivo principal de la plataforma de minado sería la estimación de conocimiento latente aplicando los algoritmos anteriormente planteados en el contexto de grandes volúmenes de datos. Por lo que se propone obtener un componente que permita aplicar el algoritmo IRT para la estimación de conocimiento latente sobre datos educacionales masivos.

En su estado actual IRT no es viable aplicarlo para el procesamiento masivo de datos educativos, debido a que existe un gran volumen de datos educacionales en el mundo moderno en comparación con los ambientes clásicos educacionales. Donde cada vez se requiere una mayor velocidad para el procesamiento y la obtención de resultados. Por otra parte, los datos pueden existir en diferentes formatos y necesariamente deben adaptarse a los parámetros que necesita el algoritmo para aplicar cualquiera de sus modelos logísticos y con estos estimar el rasgo o conocimiento latente, lo que haría muy lenta la obtención de una vista minable de los datos de forma adecuada para su posterior procesamiento.

Otro elemento a tener en cuenta en el rendimiento es el número de participantes en las plataformas educativas. Se puede observar el uso generalizado y global de los MOOCs, donde personas de todo el mundo pueden acceder a cursos de diferentes áreas, tanto de ciencias sociales, negocios, economía, ciencia de datos, ciencia de la computación, física, matemáticas entre otras. Un ejemplo de este tipo de plataforma lo constituye Coursera y según datos recogidos hasta el 2015, la misma alcanzó los 18 millones de usuarios de todo el mundo (Coursera, 2016). Cada uno de estos usuarios es evaluado constantemente por los instrumentos de los propios cursos, por lo que medir conocimiento para todos ellos constituye un verdadero desafío.

Si se aplicara IRT en una plataforma educativa de una institución universitaria para estimar el conocimiento latente de todos los estudiantes, donde el número de ellos podría estar en el orden de los miles. Conociendo además, que la plataforma posee un grupo de cursos con una cantidad

considerable de ítems evaluativos, ya que diariamente los editores, profesores y administradores pueden crear estos ítems. El rendimiento computacional del proceso de inferencia de conocimiento estaría condicionado por el volumen de estos datos, por lo que se requeriría de una gran cantidad de recursos de cómputo. Por otra parte, no se obtendrían los tiempos de respuestas deseados en el momento preciso.

Por lo anterior se plantea el siguiente **problema de investigación**:

¿Cómo incrementar el rendimiento computacional del algoritmo Teoría de Respuesta al Ítem para aplicarlo en la estimación de conocimiento latente sobre Datos Educacionales Masivos?

Se presenta como **objeto de estudio** el proceso de estimación de conocimiento latente sobre Datos Educacionales Masivos y como **campo de acción** la adaptación del algoritmo Teoría de Respuesta al Ítem para la estimación de conocimiento latente sobre Datos Educacionales Masivos.

El objetivo **general de esta investigación** es desarrollar una adaptación del algoritmo Teoría de Respuesta al Ítem, usando técnicas de procesamiento en paralelo y distribuido, para aumentar el rendimiento computacional en la estimación de conocimiento latente sobre Datos Educacionales Masivos.

Para darle solución al objetivo general se definen los siguientes **objetivos específicos**:

1. Establecer los principales fundamentos teóricos y metodológicos relacionados con la estimación de conocimiento latente sobre los Datos Educacionales Masivos.
2. Analizar las tecnologías, herramientas y técnicas para el procesamiento en paralelo y distribuido de Datos Educacionales Masivos.
3. Desarrollar una adaptación del algoritmo Teoría de Respuesta al Ítem, usando técnicas de procesamiento en paralelo y distribuido.
4. Validar la propuesta de solución desarrollada mediante la evaluación del rendimiento computacional del algoritmo.

Hipótesis científica: Si se desarrolla y aplica una adaptación del algoritmo Teoría de Respuesta al Ítem, usando técnicas de procesamiento en paralelo y distribuidas, aumentará el rendimiento computacional en la estimación de conocimiento latente sobre Datos Educacionales Masivos.

Operacionalización de las variables dependientes e independientes

Variable independiente: Adaptación del algoritmo Teoría de Respuesta al Ítem mediante el uso de técnicas de procesamiento en paralelo y distribuidas.

Variable dependiente:

- Rendimiento computacional

Tabla 1. Operacionalización de las variables

Variable independiente	**Dimensiones**	**Indicadores**	**Unidades de medida**
Adaptación del algoritmo Teoría de Respuesta al Ítem mediante el uso de técnicas de procesamiento en paralelo y distribuidas.	Eficacia	Capacidad de lograr resolver el problema para el cual fue diseñado sobre el entorno de Datos Educacionales Masivos.	Alta, Media, Baja
Variable dependiente	**Dimensiones**	**Indicadores**	**Unidades de medida**
Rendimiento computacional.	*Speedup (Sp)*	Es la razón entre el tiempo de ejecución del algoritmo secuencial (Ts) y el tiempo del algoritmo paralelo (Tp). Se puede calcular mediante la siguiente ecuación (Nielsen, 2016). $$Sp = Ts/Tp$$	Valor numérico continuo.
	Eficiencia (Ep)	Mide el grado de utilización de un sistema multiprocesador. Suele expresarse en tanto por 100. Se puede calcular mediante la siguiente ecuación (Nielsen, 2016). $$E(p) = S(P)/P$$	Valor numérico continuo.

Métodos de investigación:

Analítico-Sintético: Permitió descomponer el problema de investigación en varias partes; de esta forma fue más sencillo profundizar en el estudio de cada parte, para que luego fueran sintetizados en el desarrollo de la solución propuesta.

Inductivo-deductivo: Se empleó para identificar la técnica con que se desarrollará el algoritmo y la que mejor se aplica a las características del entorno donde este funcionará.

Experimento: Será utilizado para la validación y verificación de la hipótesis científica planteada.

Aporte y Novedad de la investigación:

A partir de la presente investigación se contará con un algoritmo que permitirá estimar conocimiento latente a partir del procesamiento Datos Educacionales Masivos. Sus características permitirán ser utilizado para la mejora de la evaluación y los instrumentos evaluativos, predecir resultados sobre grandes grupos de estudiantes frente a un determinado instrumento evaluativo.

Organización de las tesis:

El documento se encuentra organizado en tres capítulos de la forma siguiente:

Capítulo 1 Preliminares. En este capítulo se expresan los elementos fundamentales relacionados con la descripción del algoritmo IRT para la estimación de conocimiento latente. Se realiza un análisis de las diferentes herramientas y algoritmos para el procesamiento en paralelo de los datos y motores de procesamiento. Se caracterizan la variables necesarias requeridas en el proceso de estimación y la obtención de la vista minable de los datos mediante la estimación de parámetros necesarios que sirven de entrada al algoritmo.

Capítulo 2 Propuesta de solución. En este capítulo se expresan los elementos fundamentales a tener en cuenta para la estimación de conocimiento latente. Se desarrolla un algoritmo basado en el modelo que ofrece IRT en conjunto con técnicas de procesamiento en paralelo y distribuido.

Capítulo 3 Validación de la propuesta. En este capítulo se realiza la evaluación de la calidad del algoritmo desarrollado. Además se realiza un análisis del costo computacional del algoritmo desarrollado en comparación con los costos computacionales que se obtendrían si se aplica IRT en su estado original.

Finalmente se presentan las Conclusiones, Recomendaciones, el Glosario de Términos y las Referencias Bibliográficas.

CAPÍTULO 1. PRELIMINARES

En este capítulo se brinda una panorámica general sobre el proceso de estimación del conocimiento latente aplicando el algoritmo IRT y los diferentes modelos logísticos de estimación. Se analizan los diferentes métodos estadísticos para realizar el proceso de estimación de parámetros de los ítems a partir de los resultados obtenidos por los examinados mediante el uso de estimadores de máxima verosimilitud. Se realiza un estudio de soluciones similares, donde se aplican varios paradigmas de computación de alto rendimiento y se comparan las mismas midiendo su costo temporal. Posteriormente se realiza un estudio de las principales herramientas para el procesamiento masivo de datos y se selecciona la más adecuada para el desarrollo de la propuesta de solución.

1.1 Modelo IRT para la estimación de conocimiento latente y modelos logísticos de estimación

El modelo IRT surge para solventar algunas limitaciones que tenía TCT, sus inicios datan alrededor de 1960 y fue planteado por George Rasch, donde se lograron solventar deficiencias e interrogantes del viejo modelo y se basa en dos supuestos fundamentales (Pérez Gil, 2011):

- El atributo que se desea medir puede representarse en una única dimensión en la que se situarían conjuntamente las personas y los ítems.
- El nivel de una persona en el atributo y la dificultad del ítem determina la probabilidad de que la respuesta sea correcta.

Bajo estos supuestos Rasch planteó una ecuación logística que permite estimar la probabilidad de que un sujeto responda correctamente un ítem, denominado también modelo de un parámetro, pues solo tiene en cuenta la dificultad del ítem, partiendo siempre del nivel de conocimiento o rasgo latente del individuo.

El modelo logístico de un parámetro se plantea de la siguiente forma (L. Thorpe & Favia, 2012):

$$\boldsymbol{P(X_i = 1|\theta) = (1 + e^{-(\theta - \beta_i)})^{-1}} \quad [1]$$

Donde β_i indica la dificultad del ítem i,θ indica el nivel de habilidad del individuo.

Este modelo no quedó limitado a un parámetro, pues en el contexto de la evolución de los test intervienen otras variables que no se pueden despreciar en la medición, tales como: la probabilidad de adivinar un ítem o contestar correctamente al azar y el índice de discriminación o discriminador. Estos nuevos modelos fueron propuestos por Birnbaum y Fred Lord; los cuales

logran un mejor tratamiento matemático y facilitan el desarrollo de nuevos métodos de estimación de parámetros (Pérez Gil, 2011).

El modelo de dos parámetros se plantea de la siguiente manera (L. Thorpe & Favia, 2012):

$$\boldsymbol{P(X_i = 1|\theta) = (1 + e^{-a_i(\theta - \beta_i)})^{-1}} \quad [2]$$

Donde se mantiene la dificultad del ítem β_i y la habilidad del examinado θ. Además se incluye el parámetro discriminador a_i, el cual representa la magnitud de cambio en la probabilidad de acertar el ítem i, conforme varía el nivel de habilidad del individuo; su valor es proporcional a la pendiente de la recta tangente a la Curva Característica del Ítem (CCI) en su punto de inflexión (Pérez Gil, 2011) (Ver figura 1. Curva Característica del Ítem para el modelo de tres parámetros).

El modelo de tres parámetros se plantea de la siguiente manera (L. Thorpe & Favia, 2012):

$$\boldsymbol{P(X_i = 1|\theta) = c_i + (1 - c_i)(1 + e^{-a_i(\theta - \beta_i)})^{-1}} \quad [3]$$

Donde se tiene además el parámetro de coeficiente de azar o pseudoazar c_i, indicando la probabilidad de contestar correctamente al azar el ítem i.

Con las ecuaciones logísticas anteriores y partiendo de los parámetros de un ítem, estimados inicialmente, se puede obtener la CCI, donde en las ordenadas se tendrá la probabilidad de contestar correctamente el ítem conforme varíe el nivel de habilidad del individuo (F. B. Baker, 2001).

Se ha construido la siguiente CCI partiendo de los parámetros: $a = 1, \beta = 1.5, c = 0.2$.

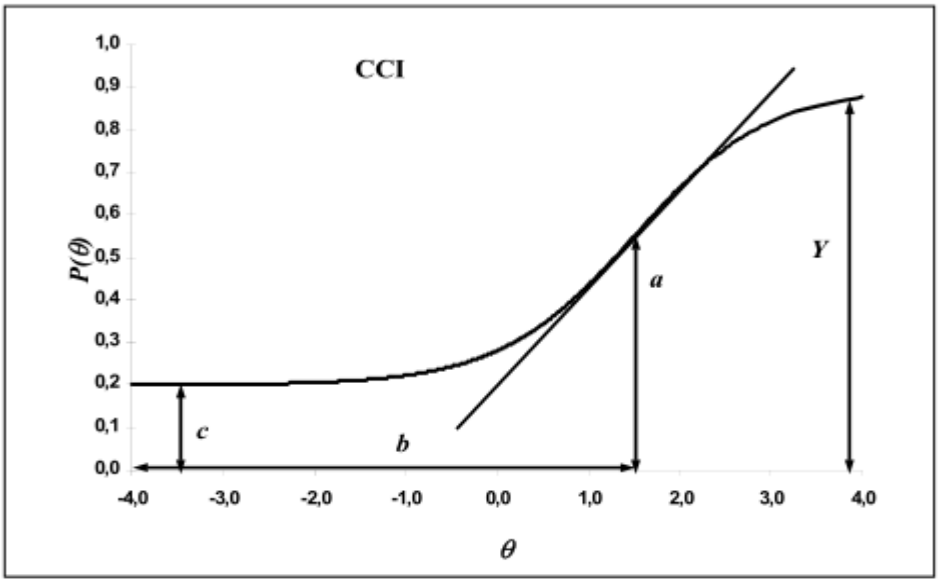


Figura 1. Curva Característica del Ítem para el modelo de tres parámetros

Se puede notar en la CCI que a partir de un punto donde aumenta el nivel de habilidad (punto de inflexión de la curva), la probabilidad de acertar dicho ítem crece considerablemente (Pérez Gil, 2011).

1.2 Estimación de la habilidad usando Teoría de Respuesta al Ítem

La estimación del nivel de habilidad de un individuo a partir de los datos dicotómicos obtenidos por un test es una parte fundamental en IRT. Conociendo los parámetros de cada ítem y calculando las probabilidades de responder correctamente, por cualquiera de los modelos anteriores; se puede estimar el nivel de habilidad de un individuo, basándose en procedimientos de máxima verosimilitud. Esto se realiza mediante un proceso iterativo que comienza con un valor de habilidad inicial determinado a priori y un vector de respuestas dicotómico (Natarajan, 2009).

La ecuación de estimación que se aplica en el algoritmo secuencial queda de la siguiente forma:

$$\boldsymbol{\theta_{s+1}} = \boldsymbol{\theta_s} + \sum_{i=1}^{n} \boldsymbol{a_i}\,[\boldsymbol{u_i} - \boldsymbol{P_i}(\boldsymbol{\theta_s})] \Big/ \sum_{i=1}^{n} \boldsymbol{a_i}^2 \boldsymbol{P_i}(\boldsymbol{\theta_s})\boldsymbol{Q_i}(\boldsymbol{\theta_s}) \qquad [4]$$

Donde θ_s es el nivel de habilidad de un individuo en la iteración s, $P_i(\theta_s)$ es la probabilidad de responder correctamente un ítem bajo su CCI y $Q_i(\theta_s)$ es la probabilidad de no responder correctamente el ítem bajo su CCI, se puede calcular mediante $1 - P_i(\theta_s)$ (F. B. Baker, 2001). Por otra parte se tiene en la ecuación la variable u_i que representa el i-ésimo valor dicotómico del vector de respuesta del individuo sobre el ítem i.

Para ejecutar el procedimiento de estimación aplicando la ecuación 4, inicialmente θ_0 se le otorga un valor arbitrario, que según la literatura puede ser uno. Luego se calcula la probabilidad de responder correctamente cada uno de los N ítems aplicando la ecuación logística correspondiente al modelo dado por la CCI (Ver ecuaciones 1, 2 y 3) y se resuelven las dos expresiones de suma. En cada iteración se actualiza θ_{s+1}, para aplicarlo a las próximas iteraciones como estimación anterior. El procedimiento concluye cuando se alcance el error estándar deseado o no ocurran cambios significativos en los valores estimados (Natarajan, 2009).

Para el cálculo del error estándar, como otro elemento a tener en cuenta en la condición de parada del procedimiento, se define la siguiente ecuación (Natarajan, 2009):

$$\boldsymbol{SE}(\boldsymbol{\theta_s}) = \boldsymbol{1} \Big/ \sqrt{\sum_{i=1}^{n} \boldsymbol{a_i}^2 \boldsymbol{P_i}(\boldsymbol{\theta_s})\boldsymbol{Q_i}(\boldsymbol{\theta_s})} \qquad [5]$$

El error estándar representa una medida de la variabilidad de θ alrededor del valor del parámetro desconocido del examinado, que es nivel de habilidad (Natarajan, 2009).

A partir del modelo anterior, para la estimación de habilidad θ de un individuo dado su vector de respuesta dicotómico $U = (u_1, u_2, \dots, u_N)$, el método es el siguiente:

Algoritmo 1. Estimación de habilidad de un individuo

<table>
<tr><td>Entrada</td><td>

Sea $U = (u_1, u_2, \dots, u_N)$ vector de respuesta dicotómico donde $u_i \epsilon \{0,1\}$ el cual representa la respuesta al ítem i.
Sea $\theta_0 = 1$ habilidad inicial a priori.
Sea $I = (\delta_1, \delta_2, \dots, \delta_N)$ vector de ítems con parámetros conocidos donde $\delta_i = \{a, b, c\}$ representa el conjunto de parámetros de un ítem i.
Sea ξ el error estándar en la estimación.
Sea Δ el cambio mínimo en la estimación.
Sea ϕ cantidad máxima de iteraciones.

</td></tr>
<tr><td>Salida</td><td>

θ habilidad del individuo.

</td></tr>
<tr><td colspan="2">Pasos:
1. Inicializar $s = 0$, $e = 0$
2. Hacer
a. $\theta_{s+1} = \theta_s + \left. \sum_{i=1}^{n} a_i \left[u_i - P_i(\theta_s)\right] \right/ \sum_{i=1}^{n} {a_i}^2 P_i(\theta_s) Q_i(\theta_s)$
b. $s = s + 1$
c. $e = \left. 1 \right/ \sqrt{\sum_{i=1}^{n} {a_i}^2 P_i(\theta_s) Q_i(\theta_s)}$
Mientras $|\theta_s - \theta_{s-1}| > \Delta$ y $e > \xi$ y $s < \phi$</td></tr>
</table>

Con este algoritmo se puede estimar el conocimiento latente de un individuo a partir de un vector de respuestas dicotómico que se obtiene sobre un grupo determinado de ítems, utilizando ítems de uno hasta tres parámetros. IRT como modelo de medición, ofrece beneficios para plataformas que poseen un banco de ítems evaluativos. Pues mediante la calibración de test que se realiza aplicando las ecuaciones 1, 2 y 3 se pueden mejorar los instrumentos evaluativos, pudiéndose aplicar en un sistema evaluador exámenes que se adaptan dinámicamente (F. Fries et al., 2014).

1.3 Estimación de parámetros

Un problema fundamental para el desarrollo de la propuesta es el procesamiento de los vectores de respuestas dicotómicos que se obtienen con las respuestas observadas de los examinados sobre los ítems. Para a partir de estos, estimar los parámetros que poseen los ítems evaluativos según los modelos logísticos elegidos. Una vez estimados, se puede inferir el conocimiento latente que poseen los individuos evaluados y su probabilidad de responder correctamente frente a un ítem determinado. Para lograr dicho proceso existen métodos estadísticos adaptados a IRT, los cuales realizan una estimación a partir de las respuestas observadas (Melià, 2017).

Los métodos estadísticos existentes para la estimación de parámetros son los siguientes (Melià, 2017):

- **Método de Máxima Verosimilitud Conjunta (JML, siglas en inglés)**. Es un método propuesto por Birnbaum en 1968, se basa en un procedimiento iterativo que permite estimar los parámetros de un ítem y de los examinados simultáneamente maximizando la función de verosimilitud para ambos elementos. Esta técnica de estimación puede ser inconsistente, lo que significa que para el modelo IRT no importan cuantos examinados estén en la muestra pues los resultados pueden estar sesgados (Johnson, 2007).
- **Estimación de Máxima Verosimilitud Condicional (CML, siglas en inglés)**. Este método es apropiado cuando se dispone de estadísticos suficientes. Un estadístico suficiente contiene toda la información muestral sobre el parámetro y reducen la dimensionalidad de los datos. Los estadísticos se obtienen a partir de una familia exponencial de distribuciones, tales como: Binomial, Bernouilli, Normal, Poison y Exponencial. Esta técnica de estimación solo se debe usar para estimar parámetros en el modelo de Rasch donde solo se tiene en cuenta la dificultad del ítem (Natesan, Nandakumar, Minka, & Rubright, 2016).
- **Máxima Verosimilitud Marginal (MML, siglas en inglés)**. Este método resulta muy adecuado cuando se desea estimar parámetros para cualquiera de los tres modelos logísticos, donde estos no tienen consistencia estructural y el número de casos aumenta. Los desarrolladores de este método (Bock y Lieberman, 1970) lograron integrarlo con el método Expectation-Maximization (EM) para que este funcionara correctamente para test evaluativos con un considerable número de ítems. Esta técnica es una de las más utilizadas en IRT (Chanjin, Xiangbin, Shaoyang, & Zhengguang, 2018). Este proceso se realiza al integrar la función de verosimilitud con respecto a la distribución normal (la distribución de los parámetros de las personas) y maximiza la verosimilitud con respecto a los parámetros de los ítems (Natesan, et al., 2016).

Existen otros estimadores, tales como los estimadores Bayesianos. Estos se basan en el teorema de Bayes, aplicado en el cálculo de probabilidades condicionales. Los mismos incorporan al proceso de estimación toda la información disponible a priori sobre la distribución de los parámetros desconocidos. La función se combina con las probabilidades de la función de verosimilitud que utiliza datos muestrales. La aplicación del teorema de Bayes produce una distribución de probabilidades a posteriori que es proporcional al producto de la función de verosimilitud y la distribución de probabilidad a priori. Las inferencias sobre los parámetros desconocidos se basan en esta distribución posterior (Melià, 2017). Estos métodos son alternativos a los anteriormente expuestos. La base estadística fundamental que distinguen los mismos son los estimadores de máxima verosimilitud (Aßmann, Gaasch, Pohl, & Carstensen, 2015).

1.3.1 Estimadores de máxima verosimilitud

Existen varias técnicas estadísticas que permiten estimar un determinado valor a partir de observaciones realizadas sobre una muestra, teniendo una función que describa los datos observados, también llamada función de densidad. Una técnica efectiva a utilizar por las características de los datos es el método de máxima verosimilitud. Esta técnica se enfoca en buscar o estimar un valor o vector de valores que haga máxima la probabilidad de obtener la muestra observada (Barcelona, 2017).

Para describir este concepto de una manera formal se plantea lo siguiente, para una mayor comprensión se hace necesario aclarar que la notación θ no representa conocimiento latente, solamente se utiliza como parte de la explicación del concepto (Jach & Alonso, 2017):

Sea $X_1, \dots, X_n$ una muestra aleatoria (no necesariamente simple) de una población X con función de probabilidad P_θ (o con función de densidad f_θ). Para cada muestra particular $(\mathrm{x}_1, \dots, \mathrm{x}_\mathrm{n})$, la función de verosimilitud se define como la función de probabilidad (o de densidad) conjunta de $(X_1, \dots, X_n)$ evaluada en $(x_1, \dots, x_n)$.

Dónde:

$$\mathbf{L}(\boldsymbol{\theta}) = \mathbf{L}(\boldsymbol{x_1}, \dots, \boldsymbol{x_n}\,;\, \boldsymbol{\theta}) = \boldsymbol{P_\theta}\,(\boldsymbol{X_1} = \mathbf{x_1}, \dots, \boldsymbol{X_n} = \boldsymbol{x_n}),\ \text{si X es un valor discreto} \quad [6]$$

$$\mathbf{L}(\boldsymbol{\theta}) = \mathbf{L}\,(\boldsymbol{x_1}, \dots, \boldsymbol{x_n};\, \boldsymbol{\theta}) = \boldsymbol{f_\theta}(\boldsymbol{X_1} = \mathbf{x_1}, \dots, \boldsymbol{X_n} = \boldsymbol{x_n}),\ \text{si X es un valor continuo} \quad [7]$$

La notación $\mathrm{L}(\theta)$ indica que L es una función de θ y no de $(\mathrm{x}_1, \dots, \mathrm{x}_\mathrm{n})$. La notación θ puede ser un escalar o un vector ($\theta = (\theta_1, \dots, \theta_\mathrm{k})$). El subíndice θ en la función de probabilidad o de densidad indica que dicha función depende del valor del parámetro.

A partir de esto se desea obtener el valor del parámetro *θ,* para el cual se maximiza la función de verosimilitud. Por lo que se define formalmente como estimador de máxima verosimilitud, según lo siguiente (Gil, 2015):

Sea $X_1, \dots, X_n$ una muestra aleatoria de una población X con función de verosimilitud L(θ). Para cada muestra particular $(x_1, \dots, x_n)$, la **estimación de máxima verosimilitud** de θ es el valor $\hat{\theta}$MV que maximiza la verosimilitud. Es decir:

$$\mathbf{L}\,(\boldsymbol{x_1}, \dots, \boldsymbol{x_n}\,;\, \widehat{\boldsymbol{\theta}}\boldsymbol{MV}) \;=\; \mathbf{max}\;\boldsymbol{\theta}\;\mathbf{L}(\mathbf{x_1}, \dots, \mathbf{x_n};\; \boldsymbol{\theta}) \qquad [8]$$

Por tanto el estimador de máxima verosimilitud es aquel que evaluado en cada muestra particular permite obtener la estimación de máxima verosimilitud.

La ecuación 4 se obtiene a partir de la aplicación de estos estimadores, donde se desea determinar para qué valores de nivel de habilidad de N individuos se obtiene la mayor probabilidad de obtener los vectores de respuestas suministrados. Descrito lo anterior de manera formal (Hambleton & Swaminathan, 1985):

Sea $u_1, \dots, u_n$ un vector de respuesta dado por N examinados con habilidad θ_a, donde a=1,…, N; la función de verosimilitud queda (Hambleton & Swaminathan, 1985):

$$\mathbf{L}(\mathbf{u_1}, \dots, \mathbf{u_n};\, \theta) = \frac{\partial}{\partial \boldsymbol{\theta_a}}\, \boldsymbol{ln}\; \boldsymbol{L}(\mathbf{u_1}, \dots, \mathbf{u_n};\, \theta) = \sum_{i=1}^{n} \frac{\partial \mathbf{lnL}}{\partial \boldsymbol{P_{ia}}}\, \frac{\partial \boldsymbol{P_{ia}}}{\partial \boldsymbol{\theta_a}} \;=\; \mathbf{0} \qquad [9]$$

Para cada examinado las ecuaciones pueden ser resueltas de forma separada. Para ello se deriva la función logarítmica de verosimilitud tantas veces lo indique el modelo logístico elegido (Hambleton & Swaminathan, 1985).

Con el fin de realizar el proceso de estimación de los parámetros de un ítem, donde se conoce el vector de respuesta, se aplicará MML como estimador para cualquiera de los modelos logísticos de IRT. Para ello basta con tener como muestra los vectores de respuestas dicotómicos de los estudiantes que se han sometido a un test de varios ítems. Con las respuestas observadas, se estimarán los parámetros de los ítems maximizando la función de verosimilitud. La elección de este estimador para el desarrollo de la propuesta de solución se fundamenta en que resuelve las limitaciones que muestran los estimadores JML y CML. Estas limitaciones están dadas por la exactitud de la estimación y el número de parámetros a considerar respectivamente. Por otra parte, MML tiene buen rendimiento para un número considerable de ítems si se aplica en conjunto con el método EM.

1.3.2 Algoritmo secuencial de EM para el modelo IRT

EM es un algoritmo iterativo que busca estimadores de máxima verosimilitud o máxima verosimilitud a posteriori, para estimar parámetros en modelos estadísticos donde este depende de variables latentes no observadas. En cada iteración del algoritmo se ejecuta un paso de Expectación (E) y un paso de Maximización (M). Este proceso se repite hasta que se alcance un número determinado de iteraciones, se obtenga el error estándar deseado en la estimación o no ocurran ya cambios significativos en la estimación (Sehgal & Garg, 2014).

Para ello el algoritmo recibe como entrada una matriz Y de tamaño $N\ x\ J$, donde N representa el número de examinados y J el número de ítems. En la entrada $Y = (y_1, y_2, \dots, y_N)$, y_i representa el vector de respuesta dicotómico del examinado i dado por $(y_{i1}, y_{i2}, \dots, y_{iJ})$ y y_{ij} es un valor de respuesta del estudiante i sobre el ítem j (Harwell, Baker, & Zwarts, 1988).

La salida del algoritmo es un vector de ítems con parámetros estimados $I = (\delta_{1,}\delta_2, \dots, \delta_j)$ donde $\delta_j = \{t_1, t_2, \dots, t_T\}$ representa el conjunto de T parámetros del ítem j.

La estimación comienza con una distribución de probabilidades condicionales de tamaño K que representan los posibles valores de la variable latente θ_i que es la habilidad del estudiante i. Los valores de la distribución condicional están denotados por $q_k, k\epsilon[1;K]$ (Harwell, et al., 1988).

A continuación se explica cómo se procede en cada paso (Insua-Suárez, Fulgueira-Camilo, & Henry-Fuenteseca, 2015):

- **Paso de Expectación (Paso E).** En este paso se crea una función para la expectación de verosimilitud de logaritmo natural, usando la estimación actual para los parámetros.
- **Paso de Maximización (Paso M).** Se calculan los parámetros deseados del modelo maximizando la función de verosimilitud de logaritmo natural calculada con el paso E.

Para estimar parámetros de los ítems mediante el algoritmo EM se ejecutan los pasos E y M de la siguiente manera (Harwell, et al., 1988):

Paso E. Calcular los valores esperados para determinar las estadísticas suficientes y completas $n_k^{(s)},\ k\epsilon[1;K]$ y $r_{jk}^{(s)}, j\epsilon[1;\mathrm{J}], k\epsilon[1;K]$ sobre la distribución condicional a partir de datos observados en Y en la iteración s mediante las ecuaciones:

$$\boldsymbol{n_k^{(s)}} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \frac{\boldsymbol{\pi_k^{(s)}} \prod_{\boldsymbol{j=1}}^{\boldsymbol{J}} \boldsymbol{P(q_k|\delta_j^{(s)})^{y_{ij}}[1-P(q_k|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}}{\sum_{\boldsymbol{k'=1}}^{\boldsymbol{K}} \boldsymbol{\pi_{k'}^{(s)}} \prod_{\boldsymbol{j}}^{\boldsymbol{J}} \boldsymbol{P(q_{k'}|\delta_j^{(s)})^{y_{ij}}[1-P(q_{k'}|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}} \quad [10]$$

$$r_{jk}^{(s)} = \sum_{i=1}^{N} \frac{y_{ij}\pi_k^{(s)} \prod_{j=1}^{J} P(q_k|\delta_j^{(s)})^{y_{ij}}[1-P(q_k|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}{\sum_{k'=1}^{K} \pi_{k'}^{(s)} \prod_{j}^{J} P(q_{k'}|\delta_j^{(s)})^{y_{ij}}[1-P(q_{k'}|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}} \quad [11]$$

- Donde $P(q_k|\delta_j)$ representa la probabilidad de responder correctamente el ítem j y δ_j los parámetros de dicho ítem, q_k el valor de la variable latente en K valores de la distribución condicional.
- La distribución de probabilidades asociada a la variable latente k en la iteración s está denotada por $\pi_k^{(s)}, k\epsilon[1;K]$

Paso M. El paso M en la iteración s consiste en sustituir los valores esperados de las estadísticas suficientes y completas obtenidas en el **paso E** que maximizan el logaritmo de verosimilitud. Con esto, los parámetros de los ítems $\delta_j^{(s+1)}$ se estiman resolviendo el siguiente sistema de ecuaciones para obtener los valores δ_{tj}, donde $t = 1, 2, 3, \dots, T_j$ siendo T_j los parámetros de ítem j.

$$\sum_{k=1}^{K} \frac{r_{jk}^{(s)} - n_k^{(s)} P(q_k|\delta_j^{(s)})}{[1-P(q_k|\delta_j^{(s)})]P(q_k|\delta_j^{(s)})} \frac{\partial P(q_k|\delta_j^{(s)})}{\partial \delta_{tj}^{(s)}} = 0 \quad [12]$$

Para resolver el sistema de ecuaciones 12 se realiza un procedimiento numérico para buscar raíces en una función aplicando el método de Newton-Raphson (Bock & Aitkin, 1981).

Luego se actualiza distribución de probabilidades asociada a la variable latente $\pi_k^{(s)}$ de la siguiente forma:

$$\pi_k^{(s+1)} = n_k^{(s)}/N \quad [13]$$

Con los elementos anteriores el método de estimación EM queda de la siguiente forma:

Algoritmo 2. Algoritmo EM para IRT

Entrada	• Sea Y matriz de tamaño $N\,x\,J$, donde N representa el número de examinados y J el número de ítems. • Sea $I' = (\delta'_1, \delta'_2, \dots, \delta'_J)$ vector de ítems con parámetros desconocidos, donde $\delta'_i = \{t_1, t_2, \dots, t_T\}$ representa el conjunto de T parámetros de un ítem i. • Sea μ máximo cambio. • Sea ϕ cantidad máxima de iteraciones.

Salida	• Vector de ítems con parámetros estimados $I = (\delta_{1,}\delta_2, \dots, \delta_J)$.

Pasos:

1. Inicializar $s = 0$, $e = 1 +$ µ
2. Inicializar $Q = (q_1, q_2, \dots, q_K)$, vector de distribución condicional.
3. Inicializar $\pi = (\pi_1, \pi_2, \dots, \pi_K)$, vector de distribución de probabilidades asociada a la variable latente $k\epsilon[1; K]$
4. Mientras $s < \phi$ y $e >$ µ
    a. Paso E.
        i. $n_k^{(s)} = \sum_{i=1}^{N} \frac{\pi_k^{(s)} \prod_{j=1}^{J} P(q_k|\delta_j^{(s)})^{y_{ij}} [1-P(q_k|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}{\sum_{k'=1}^{K} \pi_{k'}^{(s)} \prod_{j}^{J} P(q_{k'}|\delta_j^{(s)})^{y_{ij}} [1-P(q_{k'}|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}$
        ii. $r_{jk}^{(s)} = \sum_{i=1}^{N} \frac{y_{ij}\pi_k^{(s)} \prod_{j=1}^{J} P(q_k|\delta_j^{(s)})^{y_{ij}} [1-P(q_k|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}{\sum_{k'=1}^{K} \pi_{k'}^{(s)} \prod_{j}^{J} P(q_{k'}|\delta_j^{(s)})^{y_{ij}} [1-P(q_{k'}|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}$
    b. Paso M.
        i. Resolver el sistema de ecuaciones $\sum_{k=1}^{K} \frac{r_{jk}^{(s)} - n_k^{(s)} P(q_k|\delta_j^{(s)})}{[1-P(q_k|\delta_j^{(s)})]P(q_k|\delta_j^{(s)})} \frac{\partial P(q_k|\delta_j^{(s)})}{\partial \delta_{tj}^{(s)}} = 0$
        ii. Inicializar $m_0 = |\delta_1^{(s)} - \delta_1^{(s-1)}|$
        iii. Para $j = 2, \dots, J$ hacer
            1. $m_j = \max(m_{j-1}, |\delta_j^{(s)} - \delta_j^{(s-1)}|)$
        iv. $e = m_J$
        v. $\pi_k^{(s+1)} = n_k^{(s)}/N$

Los pasos E y M se ejecutan de forma iterativa hasta que se alcance el criterio de convergencia que puede estar dado por un número de iteraciones ϕ o que el máximo cambio en la estimación de parámetros e sea inferior o igual al deseado µ.

El análisis del funcionamiento del algoritmo EM permitió identificar las dependencias que existen entre los valores que se calculan en cada uno de los pasos que se realizan en el proceso de estimación de parámetros de un ítem. Con esto se pueden identificar qué partes de este pueden ser paralelizables y cómo se dividirán los datos de entrada para su procesamiento. Se observa que en el paso E para obtener los estadísticos $n_k^{(s)}$ y $r_{jk}^{(s)}$ es posible procesar cada fila de Y por separado debido a que no existen dependencias entre cada vector de respuestas observadas para obtener los resultados en las ecuaciones 10 y 11. La suma de los resultados del cálculo para los N ítems se puede realizar aplicando un operador de reducción, por la propia propiedad

asociativa de la misma. Con esto, el resultado de los estadísticos calculados se actualizaría en una variable global. Para la solución de la ecuación 12 sucede lo anterior, ya que los parámetros pueden estimarse en una iteración para cada ítem de forma individual.

1.4 Estudio de soluciones similares

Con el principal propósito de incrementar el rendimiento computacional de los algoritmos que se emplean en sistemas para el análisis de ítems y examinados bajo IRT, se han aplicado paradigmas de Computación de Alto Rendimiento (HPC, siglas en inglés). Este paradigma se basa en el empleo de clúster de computadoras o supercomputadoras para la solución de problemas complejos que requieren de un gran número de cálculos. Para ello se ejecutan los programas en paralelo logrando un menor tiempo de ejecución o comunicación (Patsias, Rahimi, Sheng, & Rahimi, 2012).

De manera general, los paradigmas HPC que se evidencian en estas soluciones son Interfaz de Paso de Mensaje (MPI, por sus siglas en inglés), OpenMP y Dispositivo de Arquitectura Computacional Unificada (CUDA, siglas en ingles). En lo siguiente, se analizarán aplicaciones que se le ha dado a estas tecnologías para resolver las limitaciones de rendimiento al emplear el modelo IRT.

1.4.1 Computación paralela aplicada a estimadores bayesianos para el modelo IRT

En esta solución se paraleliza el proceso de estimación de parámetros para IRT, tanto para los ítems como los examinados, utilizando estimadores bayesianos basados en el método numérico Markov Chain Monte Carlo (MCMC). Este método de estimación es similar a MML y funciona con buen rendimiento para entradas de datos considerablemente grandes (Johnson, 2007). En este modelo de estimación, si se implementa secuencialmente, se deben ejecutar un gran número de iteraciones para que se alcance el criterio de convergencia, por lo que el algoritmo es computacionalmente complejo y requiere de un alto costo de tiempo para ser ejecutado sobre grades datasets (Patsias, et al., 2012).

Esta solución paraleliza el algoritmo de estimación utilizando MPI, con esta se permite estimar parámetros de ítems para el modelo de ojiva de dos parámetros (2PNO, siglas en inglés). Se mide su rendimiento con respecto a su versión secuencial y se calcula su Speedup y eficiencia. Donde se nota una considerable disminución del tiempo de ejecución para un mismo dataset en la media que aumentaban los núcleos de procesamiento (Patsias, et al., 2012).

De manera similar se aplica otra versión de implementación de este mismo algoritmo de estimación pero usando CUDA. Este framework de computación paralela permite aprovechar los núcleos de procesamiento que posee una GPU o tarjeta gráfica. Cada GPU puede contener más

de 1000 núcleos de procesamiento y son más rápidas para realizar cálculos de punto flotante. Por otra parte, proveen de un mayor ancho de banda para el acceso a memoria en comparación con los CPU de múltiples núcleos (Sheng, Welling, & Zhu, 2014).

A continuación se muestra una comparación de estas soluciones junto al algoritmo secuencial que implementa este modelo. Para las pruebas se utilizaron datasets de N x K, donde N representa el número de examinados y K el número de ítems:

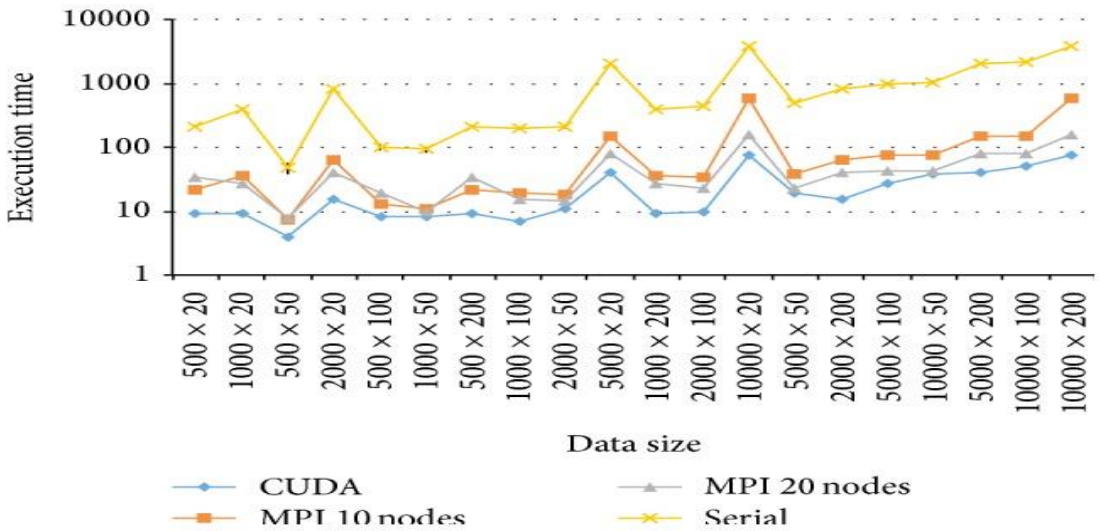


Figura 2. Tiempo de ejecución para las implementaciones de MPI, CUDA y el algoritmo serial (Sheng, et al., 2014)

Se puede observar que la variante implementada con CUDA, para todos los casos de prueba, tuvo un mejor rendimiento que la variante con MPI, debido a que en esta interviene una mayor cantidad de núcleos de procesamiento, aunque la solución que provee MPI es más escalable y permite incorporar un mayor número de nodos al clúster de procesamiento para realizar las pruebas. Por otra parte con MPI, al ser distribuido el procesamiento, intervienen las variables de tiempo de comunicación en la red lo que afecta el rendimiento considerablemente en comparación con arquitecturas de memoria compartida.

1.4.2 Paralelización del algoritmo EM utilizando OpenMP y OpenCL

Otra de las arquitecturas que se pueden emplear para el procesamiento en paralelo, es la arquitectura de memoria compartida. Esta arquitectura necesita de un procesador de varios núcleos para ejecutar tareas simultáneas. Entre los frameworks destinados para este tipo de tarea se pueden emplear OpenMP y OpenCL, donde este último permite crear aplicaciones paralelas tanto para CPU y GPU (Insua-Suárez, et al., 2015).

Se implementa una versión paralela del algoritmo EM utilizando OpenMP con un procesador de varios núcleos y con OpenCL utilizando una tarjeta gráfica NVIDIA. Para ambos casos al compararse con el algoritmo secuencial, los resultados del experimento mostraron que con OpenMP el tiempo de ejecución disminuyó en un 35% con respecto al tiempo secuencial y con OpenCL se logró disminuir en un 85% (Insua-Suárez, et al., 2015).

1.4.3 Biblioteca Psychometrics para la estimación de parámetros de un ítem

Existe una biblioteca escrita en Java llamada Psychometrics, actualmente en su versión 1.3, que incluye funcionalidades que permiten estimar parámetros a partir de los datos dicotómicos obtenidos de un grupo de estudiantes, que se han sometido a un conjunto de ítems evaluativos. Al aplicar los procedimientos que incluye la biblioteca, se pueden estimar los parámetros de un ítem para los tres modelos logísticos mediante MML. Esta biblioteca constituye un componente fundamental para el software de código abierto JMetrik que ofrece métodos de medición para el análisis de test y examinados (Andrea Gotzmann & Louise M. Bahry, 2012). Este software se aplica como solución libre para el análisis de ítems clásicos, estimación de confiabilidad, prueba de escalado, funcionamiento de ítems diferenciales, modelos de medición de Rasch y ecuaciones para IRT. La misma se encuentra disponible para las plataformas Windows, Linux y Mac OSX (J. Patrick Meyer, 2017).

La herramienta JMetrik al igual que WINSTEPS, es un sistema que implementa además métodos de estimación basados en JML para el modelo de Rasch, crédito parcial y parámetros del modelo de escala de calificación mediante un algoritmo de ajuste de curva proporcional, estos sistemas son aplicados para el análisis de individuos y test en el campo de la psicometría (J. P. Meyer, 2014).

Características de la biblioteca Psychometrics (J. Patrick Meyer, 2015)

- Posee implementaciones de estimación de parámetros mediante máxima verosimilitud marginal para los modelos de ítems de uno a cuatro parámetros y modelos de crédito parcial.
- Permite trabajar con ítems basados en modelos dicotómicos y politómicos.
- Incluye métodos de análisis clásicos de ítems, estimación de fiabilidad y funcionamiento diferencial de ítems. Característica que la hace más potente en comparación con otras herramientas para tal fin que emplean métodos no paramétricos para IRT.
- Es una biblioteca de código abierto que contiene métodos de medición para IRT, TCT y PFA.

La biblioteca Psychometrics al ser de código abierto, se puede aplicar al desarrollo de la propuesta de solución para resolver el problema de estimación de parámetros de los ítems según las características de las fuentes de datos a procesar; luego con los resultados obtenidos se realizará la inferencia de conocimiento latente aplicando el modelo de estimación que describe IRT. De esta se utilizarán algunos componentes que se encuentran disponibles en el paquete de análisis de ítems, donde se reutilizarán algunos métodos numéricos que permiten la estimación de parámetros de ítems mediante MML, cabe destacar que esta implementación del estimador ya está integrada con EM y posee una implementación paralela basada en Fork/Join.

El framework Fork/Join es una implementación que permite utilizar los múltiples procesadores disponibles en un ordenador. Su implementación se realiza de forma recursiva, donde se dividen los datos en dos porciones si el tamaño de los datos no es suficientemente pequeño, se invocan ambas porciones y se espera el resultado. Cada porción invocada se vuelve a dividir hasta que cada parte sea lo suficientemente pequeña para ser procesada directamente (ORACLE, 2017). Con esta variante se logra un buen rendimiento aunque está limitada a la memoria que posee el ordenador, ya que utiliza memoria compartida.

En los casos analizados anteriormente, se puede constatar que el uso de tarjetas gráficas para el procesamiento permite alcanzar un alto rendimiento si los datos no tienen gran tamaño y requieren de un cómputo complejo. Pero en la medida que los datos crecen se hace necesario el uso de arquitecturas que permitan el procesamiento distribuido, ya que la memoria y núcleos de procesamiento que posee un solo ordenador es muy limitada. Por lo que el uso de MPI se puede considerar como una alternativa certera aunque intervenga el tiempo de comunicación a través de la red. Para elegir el paradigma adecuado para el desarrollo de la propuesta de solución se analizarán con mayor profundidad los conceptos de computación distribuida, HPC y sus diferentes paradigmas.

1.5 Computación distribuida y computación de alto rendimiento

El desarrollo de las redes de telecomunicaciones ha permitido resolver un grupo de problemas que presentan un alto costo computacional en cuanto a capacidad de procesamiento y memoria. Para resolver problemas de computación masiva surgen los modelos de computación distribuida, donde un grupo de agentes autónomos, cada uno con capacidades de procesamiento distinto, se comunican entre sí de forma orquestada y afectan mutuamente su comportamiento. Estos sistemas permiten resolver problemas complejos con un alto rendimiento y proveen de alta disponibilidad en caso de fallo de alguno de sus componentes (Attiya & Welch, 2004). Los agentes también son llamados frecuentemente, nodos, procesadores o procesos; estos pueden ser desde teléfonos inteligentes hasta potentes ordenadores. Con los agentes integrados se pueden resolver

tareas de gran complejidad mediante un sistema de alto nivel que coordina cada agente o nodo (Aroquipa, 2018).

La computación distribuida, de forma general, hace referencia a los servicios que provee un sistema distribuido. Los sistemas distribuidos cumplen con las siguientes propiedades (Aroquipa, 2018):

- Están conformados por recursos informáticos de propósito general, tanto físicos como lógicos y se le asignan dinámicamente tareas concretas.
- Los recursos están distribuidos físicamente y funcionan mediante su comunicación a través de la red.
- Hay un sistema operativo de alto nivel que unifica e integra el control de los componentes.
- El funcionamiento de los recursos físicos y lógicos se caracteriza por una autonomía coordinada.
- Los servicios pueden ser solicitados por su nombre específico, por lo que la distribución de los recursos es transparente.

Con esto se puede apreciar que la computación distribuida tiene la particularidad de ser fácil de expandir y la interacción con el usuario final que utiliza los servicios, es de manera consistente y uniforme. Esta constituye una alternativa a las costosas supercomputadoras al estar formada por una extensa red nodos interconectados entre sí. En estos sistemas no importa la ubicación de los recursos con capacidad de procesamiento, ya que estos son coordinados para cumplir una función específica con mayor velocidad de respuesta. Permiten además, disponer de manera concurrente de un equipo de ordenadores en función de darle solución a una tarea o de proveer un servicio determinado (Skala, Davidovi, Afgan, Sovic, & Sojat, 2015).

1.5.1 Computación de alto rendimiento

La computación de alto rendimiento o HPC como ya se ha mencionado, es un área que abarca los diferentes paradigmas de programación paralela, sus lenguajes de programación relacionados, las interfaces de programación de aplicaciones (API) y las herramientas de software dedicadas. Es a su vez un campo científico y técnico del estudio de las supercomputadoras (Nielsen, 2016).

La principal utilidad de HPC se enfoca en la solución de problemas de alta complejidad computacional, tales como la simulación de problemas complejos y fenómenos de la naturaleza. También se evidencia su uso en situaciones donde se hace necesario un tiempo de respuesta mínimo y se utilicen cantidades masivas de datos o grandes datasets; de ahí su importancia para el procesamiento Big Data, donde están presentes las cuatro Vs de los grandes datos masivos (Volumen, Variedad, Valor y Velocidad). Estas hacen referencia a la naturaleza variada,

heterogénea y extensa de los datos, la velocidad con la que crecen, aparecen nuevos registros de datos y su veracidad (Trifu & Ivan, 2014). De forma general HPC está concebido resolver tareas donde intrínsecamente se aplican algoritmos paralelos (Nielsen, 2016).

HPC posee varios paradigmas y áreas de estudio, hoy en día los modelos de programación más difundidos son los siguientes (Nielsen, 2016):

- **Programación con MPI.** El cual tiene cero tolerancias a fallos en el software y el hardware o errores en la red, pero que ofrece un marco de trabajo flexible.
- **Programación con MapReduce.** Incluye una infraestructura tolerante a fallos y errores de hardware y software, pero con un modelo limitado de computación paralela en comparación con MPI.

Las soluciones que aplican paradigma MPI están caracterizadas por ser más orientadas al rendimiento computacional, donde se utilizan directivas que permiten una programación distribuida más flexible. Se recomienda aplicar MPI cuando el tamaño de los datos es moderado y el cómputo es complejo (Kang, Lee, & Lee, 2015). Si el tamaño de los datos se intensifica se hace necesario el uso del paradigma MapReduce, ya que este provee de un mecanismo tolerante a fallos y de gestión de la replicación de datos. Por otra parte, este paradigma está orientado al procesamiento de cantidades masivas de datos y para ello emplea sistemas de archivos distribuidos, donde en caso de fallos o perdidas de datos, este permite recuperarlos y volver a ejecutar la tarea que estaba realizando de forma transparente al desarrollador (Reyes-Ortiz, Oneto, & Anguita, 2015). Además ofrece una solución para el procesamiento distribuido de los datos en un entorno no dedicado, lo que significa que puede ser desplegado en estaciones de trabajo conectadas a la red aprovechando los recursos de memoria y procesamiento que no se estén utilizando por los usuarios, lo que lo hace una alternativa muy económica y permite convertir en un gran clúster cualquier laboratorio de computadoras personales.

Existen herramientas que implementan el modelo de programación MapReduce, muchas de ellas son de código abierto. Para el desarrollo de la propuesta de solución se analizarán herramientas que lo aplican, las cuales permiten el procesamiento distribuido de datos masivos con alta tolerancia a fallos de diferente índole.

1.5.2 MapReduce como modelo de programación distribuida

El paradigma MapReduce es un modelo de programación que permite paralelizar procesos en dos fases. La fase de mapeo donde se realiza la recogida de los datos de entrada produciendo una lista de pares clave/valor. Estos pares son agrupados por clave y pasados a la función reduce que

se encarga de procesarlos y generar un resultado agrupado de los mismos (Dean & Ghemawat, 2004).

El funcionamiento de este paradigma distribuido se basa en generar un conjunto de pares formados por clave/valor a partir de una entrada y luego mesclar los resultados obtenidos en conjuntos más pequeños, cada operación de mapeo y reducción se puede ejecutar de forma distribuida y paralela en múltiples nodos, logrando un potente procesamiento masivo de datos (Dean & Ghemawat, 2004).

El procedimiento de generar tuplas en pares clave/valor se realiza haciendo uso de la función Map, disponible en la biblioteca MapReduce que poseen varios frameworks para el procesamiento distribuido. Esta función de mapeo puede ser definida por el usuario, con la cual este especificará la forma en la que se mapearán los datos según las características de los mismos. Con la salida obtenida, el usuario podrá acceder a la función Reduce, la cual recibirá como entrada la salida de la función Map y la dividirá en un conjunto más pequeño de tuplas (White, 2012).

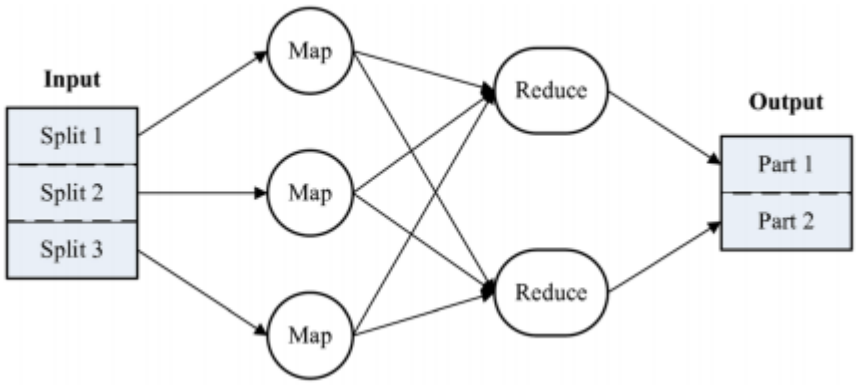


Figura 3. Modelo de programación MapReduce (Teng, 2014)

Se debe tener en cuenta que los datos entrada que recibe la función Map provienen de los sistemas Hadoop Distributed File System (HDFS), colecciones o ficheros de texto almacenados en el clúster de computadoras. Con los datos, la función revisa línea por línea el archivo de entrada y genera varios fragmentos de datos (White, 2012).

En la etapa de reducción (Reduce) se combina la producción obtenida por la etapa Map. Se procesan los pares de la forma deseada por el usuario y se genera un nuevo conjunto de producción que se almacena de la misma forma en un HDFS (White, 2012).

Existen un grupo de herramientas que permiten hacer uso de este paradigma o modelo distribuido de programación basado en funciones de mapeo y reducción, lo cual las hacen potentes para el

procesamiento masivo de los datos, al dividir datos grandes en conjuntos más pequeños. Este procesamiento en paralelo brindado por el paradigma puede ser personalizado por el usuario en consonancia con las características de los datos y la necesidad. Los clústeres de computadoras que almacenan los conjuntos de datos no transfieren datos por la red, solo se transfiere el código o instrucciones para su procesamiento. Por otra parte los resultados de procesamiento se pueden almacenar en los clústeres de computadoras utilizando un sistema de archivos distribuido, como el conocido HDFS (Guller, 2015).

1.6 Herramientas para el procesamiento de datos masivos

El uso de la computación distribuida se ha hecho masivo en varias ramas de la ciencia y la tecnología, donde cada vez se generan cúmulos masivos de datos a gran velocidad. Para procesar tanta información existen diferentes herramientas y soluciones tecnológicas que proveen modelos de programación para el procesamiento distribuido de los datos en clústeres de computadoras que se comunican a través de la red. Estas herramientas y tecnologías están categorizadas como herramientas para el análisis Big Data y se destacan por su difundido uso las siguientes: Apache Hadoop, Apache Spark, Apache Storm, Apache Hive, Apache Flink y Apache Kafka (BAOSS, 2015). Cada una de estas herramientas tiene un fin determinado para resolver problemas de minería de datos masivos, según las características de las fuentes de datos. Poseen diferentes valores de rendimiento sobre diferentes escenarios. Incluyen librerías para el aprendizaje automático y un sistema de archivos distribuido para el almacenamiento de los datos masivos (Landset, Khoshgoftaar, Richter, & Hasanin, 2015).

Para la elección de la herramienta de análisis Big Data para el desarrollo de la propuesta de solución, se realizó un estudio que permitió analizar el funcionamiento general de las mismas, sus características, funcionalidades que incorporan y su rendimiento para el procesamiento de datos masivos. De las herramientas antes mencionadas, son elegidas para el análisis y comparación Apache Hadoop, Apache Spark y Apache Flink; las cuales implementan eficientemente y extienden el modelo de programación MapReduce (Chatzigeorgakidis, Karagiorgou, Athanasiou, & Skiadopoulos, 2017).

1.6.1 Herramienta Apache Hadoop

Apache Hadoop es un framework de código abierto escrito en Java, utilizado para realizar procesamiento distribuido sobre un clúster de máquinas. Dispone de un sistema de archivos distribuido HDFS con tamaño de bloques que van desde 64 MB hasta 128 MB, pensado para trabajar sobre grandes volúmenes de datos. Dicho sistema de archivos está basado en Google File System (GFS), el mismo fue creado con el objetivo de ser distribuido, tolerante a fallos, escalable, con un intensivo acceso a datos y con alta concurrencia (White, 2012).

Apache Hadoop cuenta con tres formas para ser utilizado e instalado. Dependiendo de las necesidades, se puede optar por varios tipos de instalación o modos de funcionamiento (White, 2012):

- Un único nodo en local (single node), utilizado para hacer pruebas de concepto corriendo Hadoop en una misma máquina.
- Un clúster pseudo-distribuido para simular un clúster de varios nodos ejecutándose una misma máquina.
- Montar un clúster entre distintas máquinas (multi node) totalmente distribuido. Este modo es el utilizado en producción para una plataforma de procesamiento masivo de datos.

1.6.3 Herramienta Apache Spark

Apache Spark es un framework de código abierto para clústeres de computadoras, provee de APIs para clusterización y la aplicación de algoritmos de aprendizaje automático. Soporta procesamiento en paralelo y es tolerante a fallos. En comparación con Apache Hadoop, este framework es 100 veces más rápido en memoria y 10 veces más rápido en cuanto acceso a disco (Spark, 2017).

Características de Apache Spark (Spark, 2017)

- Es fácil de usar y se pueden aplicar los lenguajes: Java, Scala, Python y R.
- Ofrece 80 operaciones de alto nivel que facilitan la construcción de programas paralelizados.
- Combina para el acceso a los conjunto de datos fuentes en SQL, streaming en tiempo real y analítica compleja.
- La herramienta puede ejecutarse dentro del propio entorno Hadoop y acceder a los HDFS.
- Posee una gran comunidad y abundante documentación para su uso.

Esta herramienta de procesamiento distribuido por su alta velocidad de acceso de lectura/escritura en disco y procesamiento en memoria, se ha convertido en un reemplazo para Hadoop/MapReduce. Este incremento de la velocidad es un elemento clave para la realización de tareas de procesamiento masivo de datos. Esta diferencia notable en el rendimiento es debido a que las aplicaciones creadas en Spark guardan los datos en una cache de memoria, lo cual minimiza el acceso de lectura y escritura en disco. Por otra parte, las aplicaciones creadas sobre Hadoop/MapReduce consisten en una secuencia de tareas donde cada una lee los datos almacenados en disco, los procesa y luego escribe los resultados en disco. En función del tamaño de los datos a procesar, este proceso puede realizar el acceso a disco varias veces, mientras que

con el cache de los datos en memoria tal y como lo hace Spark, el acceso se realiza una sola vez. La diferencia en el rendimiento no se hace notable para un volumen poco significativo de datos, pero si la cantidad de datos crece considerablemente entonces se puede notar la diferencia (Guller, 2015). Esta ventaja hace de la herramienta Apache Spark una mejor elección para el desarrollo de la propuesta de solución.

1.6.4 Herramienta Apache Flink

Apache Flink es un marco de código abierto escrito en Java y Scala para el procesamiento de flujo distribuido que proporciona resultados precisos, incluso en el caso de datos fuera de orden o que llegan tarde. Es estable y tolerante a fallos y puede recuperarse a la perfección de estos mientras mantiene el estado de aplicación exactamente una vez. Funciona a gran escala, se ejecuta en miles de nodos con muy buen rendimiento y características de latencia (Flink, 2018b).

Flink posee muchas funciones para el manejo de datos fuera de orden, aplica el uso de Streaming windows para procesar resultados precisos en conjuntos de datos ilimitados y son habilitados por el modelo de ejecución de transmisión de Flink (Yu & Guo, 2015).

De este framework se pueden destacar las siguientes características (Flink, 2018b):

- Admite procesamiento de flujo y ventanas con semántica de tiempo del evento. El tiempo del evento hace que sea fácil calcular resultados precisos sobre transmisiones donde los eventos llegan desordenados y donde los eventos pueden llegar retrasados.
- Admite ventanas flexibles basadas en el tiempo, el recuento o las sesiones, además de las ventanas basadas en datos. Las ventanas se puede personalizar con condiciones de activación flexibles para admitir patrones de transmisión sofisticados y permiten modelar la realidad del entorno en el que se crean los datos.
- La tolerancia de fallos es liviana y permite que el sistema mantenga altas tasas de rendimiento y proporcione garantías de uniformidad una vez al mismo tiempo. Se recupera de fallas con ninguna pérdida de datos. La compensación entre confiabilidad y latencia es insignificante.
- Posee un alto rendimiento y baja latencia con indicadores superiores a Apache Storm.
- Está diseñado para ser ejecutado en grandes clústeres de computadoras y se caracteriza por su alta escalabilidad.
- Provee de un mecanismo de control de versiones, que permite actualizar aplicaciones o reprocesar datos históricos sin pérdida de estado y tiempo de inactividad mínimo.

Se puede decir que esta herramienta está centrada en el procesamiento distribuido de datos que provienen de un streaming o flujo. La misma se caracteriza por poseer una alta tolerancia de fallos

con cero perdida de datos. El procesamiento de flujo de datos que provee la herramienta, permite realizar un análisis de los mismos en tiempo real y si el usuario lo desea puede regresar a estados anteriores de los datos con poco tiempo de inactividad (Flink, 2018b). Estos elementos analizados la hacen una herramienta potente para realizar tareas de procesamiento en tiempo real, pero la plataforma de minado donde se incluirá la propuesta de solución trabajará con datos ya previamente almacenados en sistemas de archivos distribuidos HDFS.

1.6.5 Selección de la herramienta para el procesamiento de datos masivos

Para la selección de la herramienta a utilizar en el desarrollo de la propuesta de solución se analizan un grupo de características y elementos que las distinguen unas de otras. Los elementos analizados son aquellos que constituyen de alta importancia para la para la plataforma de minado y su escalabilidad, fundamentalmente la tolerancia a fallos, compatibilidad con diferentes sistemas archivos distribuidos y algoritmos de aprendizaje automático.

Tabla 4. Comparación entre las herramientas analizadas para el procesamiento distribuido de datos masivos

Características/ Herramientas	**Apache Spark** (Pentreath, 2015)	**Apache Flink** (Flink, 2018a)	**Apache Hadoop** (White, 2012)
Sistemas de archivos y almacenamiento	Compatible con los sistemas de Archivos de Hadoop: HDFS, Casandra, Amazon S3.	Utiliza los sistemas de archivos que provee Hadoop. Permite adicionar nuevas implementaciones de sistemas de archivos.	Solo incluye HDFS.
Tolerancia a fallos	Carga los datasets distribuidos en una estructura de datos tolerante a fallos RDD (Conjunto de Datos Distribuido Resistente, siglas en inglés).	Permite el trabajo con datasets ilimitados que procesa en tiempo real, provee un mecanismo de recuperación que permite cero perdida de datos, mediante la creación de puntos de control o checkpoints.	Aplica el mecanismo que ofrece su propio sistema de archivos escalable HDFS. En este sistema los ficheros son replicados varias veces y en caso de fallos no se abortan los procesos de cálculo.
Algoritmos de aprendizaje automático	Incluye una extensa librería de aprendizaje automático, para resolver problemas de clasificación, regresión, clustering,	Posee algoritmos de aprendizaje supervisado y no supervisado, clustering, regresión lineal y recomendación.	No incluye algoritmos de aprendizaje automático. Solo la implementación del modelo MapReduce.

	recomendación, reducción de dimensionalidad y detección de anomalías.		
Disponibilidad de APIs	Disponible para Java, Scala, Python y R.	Disponible para Java y Scala.	Disponible para Java.
Formas de despliegue	• Un solo nodo • Modo Standalone • Apache YARN • Apache Mesos • Amazon EC2	• Modo Standalone • Apache YARN • Apache Mesos • Docker • MapR	• Un solo nodo • Modo pseudo-distribuido • Apache YARN

A partir del análisis anterior, se elige como herramienta de procesamiento de datos masivos al framework Apache Spark, debido a su mayor rendimiento con respecto a Apache Hadoop/MapReduce para el procesamiento de datos masivos, al ser este 100 veces más rápido si los datos a procesar se almacenan en memoria y 10 veces más rápido si los mismos se almacenan en disco. Apache Spark incluye además librerías con algoritmos de aprendizaje automático que resuelven un gran número de tareas para la predicción y la descripción en la minería de datos. Provee de APIs para desarrollar aplicaciones en modo Standalone, o sea que se ejecutan de forma autónoma mediante un servidor interno. Las APIs están disponibles para los lenguajes Java, R, Python y Scala, este último es el más recomendado. La documentación de la herramienta es abundante y permite integrarse con Apache Hadoop, utilizando además su propio sistema de archivos distribuido. Por lo anterior el desarrollo de la propuesta de solución aplicará el uso de la herramienta.

1.7 Propuesta de solución como un servicio

Las tecnologías descritas y seleccionadas anteriormente para la propuesta de solución permiten una correcta y efectiva integración con tecnologías basadas en servicios REST (Transferencia de Estado Representacional, por sus siglas en inglés). Esto hará posible que otras aplicaciones o sistemas puedan interactuar con la plataforma de minado de Datos Educacionales Masivos y consumir servicios que esta tendrá disponible. Los servicios podrán ser invocados de forma remota mediante el envío de peticiones y el sistema dará una respuesta en un formato que podrá ser procesable por aplicaciones clientes. A continuación se describen las tecnologías que serán aplicadas para la capa de servicios de la propuesta de solución.

1.7.1 Servicios basados en tecnología REST

Los servicios basados en REST constituyen hoy en día una alternativa más simple a SOAP (Simple Object Access Protocol) y los servicios basados en WSDL (Lenguaje de Descripción de Servicios Web, por sus siglas en inglés). Se pueden encontrar varios casos de éxitos en muchas empresas que han migrado sus servicios a estas tecnologías, tales como: Yahoo, Facebook y Google. Cabe destacar que inicialmente fue presentado por Roy Fielding en el año 2000 en una conferencia de la Universidad de California, donde presentaba principios arquitectónicos de software para usar a la web como una plataforma de servicios (Rodriguez, 2015).

REST define un conjunto de principios arquitectónicos por los cuales se diseñan servicios web haciendo énfasis en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes (Newman, 2015).

Características de los servicios basados en REST (Rodriguez, 2015)

- **Utiliza de manera explícita métodos HTTP.** De los métodos HTTP se definen los siguientes: POST para crear un recurso en el servidor, GET para obtener un recurso, PUT para cambiar el estado de un recurso o actualizarlo y DELETE para eliminar un recurso (Newman, 2015). Constituye una característica clave de los servicios web REST el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616, donde cada método tiene una función específica.
- **No mantiene estado.** Necesitan escalar para poder satisfacer una demanda en constante crecimiento. Se usan clústeres de servidores con balanceadores de carga y alta disponibilidad, proxies, y gateways para conformar una topología de servicios, que permita transferir peticiones de un equipo a otro para disminuir el tiempo total de respuesta de una invocación al servicio web.
- **URIs con forma de directorios.** Permite crear URIs para los servicios de una forma jerárquica y organizada permitiendo mejor documentación de los servicios y que se utilicen estos de manera intuitiva.
- **Transfiere XML (Lenguaje de Marcado Extensible, siglas en inglés), JavaScript Object Notation (JSON, por su siglas en inglés), o ambos.** Se pueden usar para atender las diferentes peticiones invocadas mediante el envío de una respuesta en formato XML o JSON. Están notaciones hacen extensibles su uso y en la práctica permiten fácilmente consumir los servicios sin importar los lenguajes en las que están construidas las aplicaciones clientes.

1.7.2 Frameworks para la creación de servicios basados en REST

Para la selección de la herramienta a utilizar para la creación de servicios REST, se tiene en cuenta que la plataforma que se desarrollará para el procesamiento de datos masivos será un sistema distribuido que empleará tecnologías y herramientas construidas en Java; principalmente se evidencia esto en la herramienta para el procesamiento Big Data que se empleará. Por lo que lo más adecuado es emplear tecnologías con similares características y así lograr una mayor compatibilidad entre los componentes del sistema distribuido.

Para el lenguaje Java existen varios frameworks que permiten el desarrollo de servicios basados en REST, tales como: Dropwizard (Dropwizard, 2017), Play Framework (Richard-Foy, 2014), RESTEasy (Burke, 2014) y Spring Boot (Newman, 2015). Los cuales trabajan de forma similar en el proceso de creación de servicios basados en REST y algunos de estos incluyen de forma embebida servidores de aplicaciones para Java web. De estos frameworks, se puede observar que implementan servicios REST basados en una alta modularidad, poseen buen balance de carga, son fáciles de utilizar y poseen abundante documentación y soporte (Allamaraju, 2010).

1.7.3 Framework Spring Boot para la creación de servicios REST

Spring Boot es un framework que posee las bibliotecas necesarias para la creación de aplicaciones ejecutables basadas en Spring (Framework Java basado en Modelo-Vista-Controlador). Los componentes que posee permiten simplificar los pasos de la selección de las dependencias necesarias para el proyecto a desarrollar y su despliegue en el servidor. No requiere de configuración usando archivos en XML y permite la creación de servicios REST en Java de manera fácil, centrando al desarrollador en los elementos críticos de desarrollo de su aplicación. Proporciona además, una gama de características no funcionales que son comunes a grandes clases de proyectos (servidores integrados, seguridad, métricas, controles de estado y configuración externalizada) (Turnquist, 2014).

Para el desarrollo de la propuesta de solución se utilizará Spring Boot debido a las características de la plataforma de procesamiento que se va a desarrollar, ya que será un sistema distribuido soportado en herramientas y tecnologías basadas en lenguaje Java. Esta selección previa fue analizada con anterioridad por el equipo de investigación del grupo GITAE-MDEM. Por otra parte, Spring Boot permite la creación de aplicaciones que se ejecutan en modo Standalone al poseer de manera embebida un servidor de aplicaciones para ejecutarse de forma autónoma con solo invocar el producto desplegado usando el comando *“java –jar <producto>.jar”.* El uso de anotaciones para la definición de los servicios y los controladores REST facilita el desarrollo rápido de los servicios aplicando un bajo acoplamiento, esto permite separar las diferentes funcionalidades y descentralizar la solución, permitiendo el desarrollo de Microservicios (Antonov,

2015). Admite seguridad basada en OAuth2 y otros mecanismos para la autorización y control de seguridad de las aplicaciones web o desktop, tales como: http-basic y JSON Web Token. Por otra parte provee de forma completa los beneficios asociados al framework de desarrollo Spring MVC.

1.7.4 API de documentación de servicios Spring Fox

Para generar la documentación de la API de servicios REST se utilizará Spring Fox la cual permite documentar de forma rápida y sencilla servicios construidos con Spring Boot. Esta se basa fundamentalmente en el uso de anotaciones para definir de forma completa cada detalle en la documentación de una determinada funcionalidad definida como un servicio. Trabaja sobre la propia configuración establecida por el framework Spring MVC y se integra fácilmente a los mecanismos de seguridad empleados. Se basa en una semántica sencilla y provee de una interfaz web extensible y personalizable para publicar la documentación de los servicios a través de Swagger (SpringFox, 2017).

1.8 Conclusiones del capítulo

A partir de los elementos expuestos se arriban a las siguientes conclusiones:

- El estudio del algoritmo Teoría de Respuesta al Ítem permitió caracterizar el proceso de estimación de conocimiento latente, aplicando los diferentes modelos logísticos existentes. Con esto se identificaron los elementos necesarios que se necesitan para llevar a cabo este proceso, formalizando además el algoritmo secuencial.
- El estudio de métodos estadísticos para la estimación de parámetros a partir de las respuestas observadas, permitió elegir el estimador MML combinado con EM para realizar la estimación de parámetros de ítems con un buen rendimiento cuando los datos crecen considerablemente.
- El análisis de varios modelos de programación distribuida permitió identificar a MapReduce como un modelo adecuado para el desarrollo de la propuesta de solución al ser altamente tolerante a fallos.
- El estudio de herramientas para el procesamiento distribuido permitió elegir al framework Apache Spark para el desarrollo de la propuesta de solución, debido a los mecanismos de tolerancia a fallos y su velocidad en comparación con Apache Hadoop.
- El uso de Spring Boot como framework para la creación de servicios basados en REST permitirá que la solución sea interoperable con otras aplicaciones, posibilitando que la propuesta procese los datos de forma remota mediante el envío de peticiones por HTTP.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

En el presente capítulo se analiza la propuesta de una adaptación del algoritmo IRT a un modelo de programación distribuida para la estimación de conocimiento latente sobre datos educacionales masivos. Esta propuesta se sustenta a partir del cálculo del nivel de habilidad de un individuo propuesto en el modelo IRT y el proceso de estimación de parámetros mediante MML combinado con EM. Los procedimientos matemáticos son adaptados para ser aplicados en un contexto distribuido. Se formaliza el problema a resolver y se describe la adaptación del algoritmo que permite resolver dicho problema.

2.1 Modelación del problema

Para dar paso a la modelación del problema se asumen las siguientes premisas:

- Los datos de entrada del algoritmo se tomarán a partir de datasets formados por una matriz Y de tamaño $N\ x\ J$, donde N representa el número de examinados y J el número de ítems.
- Cada $y_i \epsilon Y$, con $i = 1, 2, \dots, N$; representa el vector de respuesta dicotómico del examinado i dado por $(y_{i1}, y_{i2}, \dots, y_{iJ})$ y y_{ij} es un valor de respuesta del estudiante i sobre el ítem j.
- Cada y_i está formado por J elementos separados por coma donde cada valor de respuesta se representa como: $y_{ij} \in \{0,1\}, \forall\ i\ \in [1, N]$.
- El vector $I = (\delta_1, \delta_2, \dots, \delta_J)$ representa los parámetros de los J ítems, donde $\delta_i = \{t_1, t_2, \dots, t_T\}$ representa el conjunto de T parámetros de un ítem j.

Una vez establecido lo anterior se puede plantear como **Formulación de problema**:

Sea S un conjunto de datos de entrada formado por la matriz Y, se desea obtener un conjunto de salida R formado por los pares $(\theta_i; SE(\theta_i))$. Donde θ_i representa el conocimiento latente del i-ésimo examinado y $SE(\theta_i)$ el error estándar en la estimación de θ_i.

2.2 Estructura general de la propuesta

La solución propuesta está basada en la ecuación de estimación de habilidad descrita en el acápite 1.2, donde se aplican las ecuaciones logísticas definidas por el modelo IRT mediante un proceso iterativo a partir del vector de respuesta que provee el examinado y su habilidad inicial. Esta ecuación se aplica cuando los parámetros de los ítems se conocen. Para dar paso a este

proceso, se deben primero estimar los parámetros de los ítems mediante MML combinado con EM.

A continuación se muestra un esquema general de la propuesta:

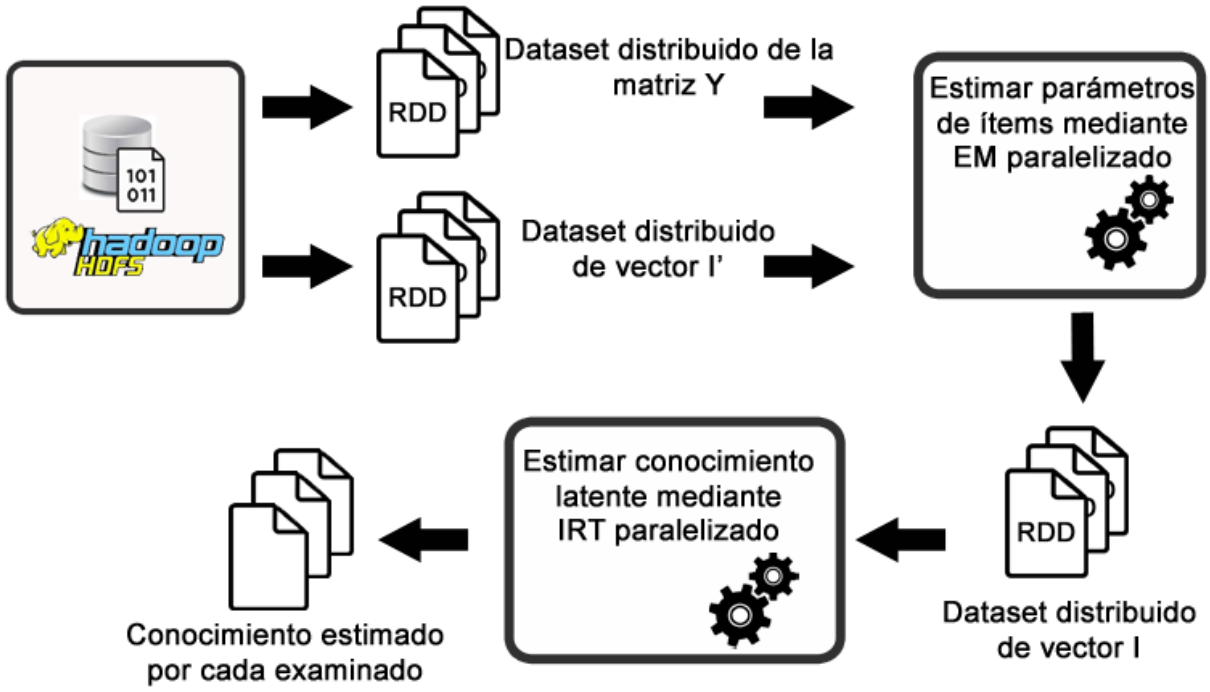


Figura 4. Esquema general de la propuesta de solución (Fuente: Elaboración propia)

El algoritmo se integrará a la plataforma de minado de datos educacionales masivos, que a su vez ejecutará otros algoritmos de estimación distribuidos.

Para su funcionamiento, los datos de entrada se toman de sistemas de archivos distribuidos HDFS que contienen los datasets. Se procede a estimar los parámetros de un ítem a partir de las respuestas observadas por los examinados y luego se estima el conocimiento latente para cada examinado. Ambos procesos se ejecutan de forma distribuida.

Como resultado de la ejecución del algoritmo se genera un fichero que se almacena en uno de los nodos físicos de la plataforma con los valores de estimación de conocimiento por cada estudiante y su correspondiente error estándar en la estimación.

2.3 Estructura del algoritmo propuesto

De forma general el algoritmo está compuesto por tres pasos fundamentales. El primer paso consiste en crear los datasets distribuidos del vector de respuestas e ítems con parámetros desconocidos, utilizando la estructura RDD que provee Apache Spark para el procesamiento

distribuido de los datos. El segundo paso consiste en estimar los parámetros de los ítems a partir de las respuestas observadas por cada examinado mediante el método de MML combinado con EM. Por último, el tercer paso consiste en estimar la habilidad de los examinados aplicando la ecuación de estimación de habilidad explicada en el acápite 1.2.

A continuación se muestran los pasos lógicos para la construcción del algoritmo:

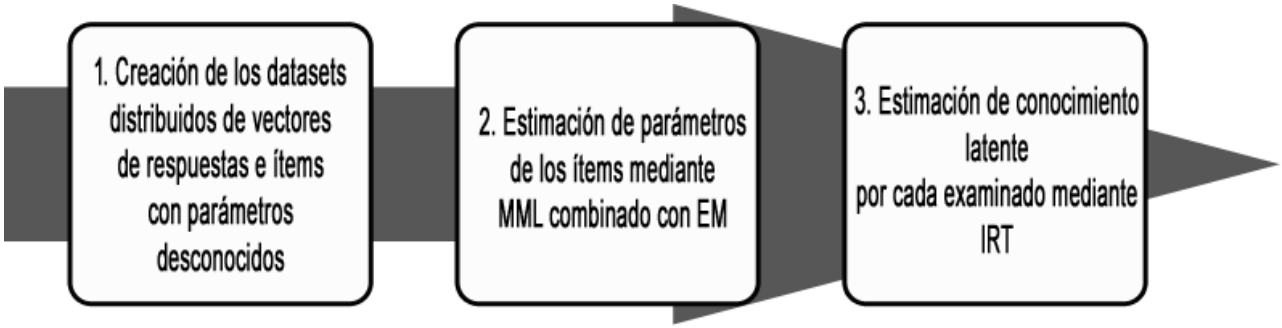


Figura 5. Pasos lógicos para la construcción del algoritmo (Fuente: Elaboración propia)

Los pasos del algoritmo se ejecutan secuencialmente, cada paso se procesa de forma distribuida por los clústeres de la propuesta de solución. Los pasos 2 y 3 son ejecutados hasta que se alcance el criterio de parada.

2.3.1 Elementos principales del algoritmo propuesto

El algoritmo propuesto utiliza los siguientes elementos para su funcionamiento:

- Procedimiento para la obtención de los datasets distribuidos del vector de respuestas e ítems con parámetros desconocidos a partir del procesamiento del dataset en formato de texto. Este dataset contiene los vectores de respuestas dicotómicos por cada examinado y se almacena en un HDFS.
- Estimación de parámetros de los ítems. En este paso se aplica una paralelización del algoritmo EM para el modelo IRT. El procedimiento está basado en el Algoritmo 2 explicado en el sub-acápite 1.3.2. Se utilizan en este paso algunos componentes provistos por la biblioteca Psychometrics. Esta técnica fue adaptada para ser procesada de forma distribuida mediante el uso de la estructura RDD que provee Apache Spark para manipular datos. Se realizan las transformaciones y acciones necesarias para el procesamiento del vector de respuestas y la actualización de los parámetros de los ítems.
- Estimación de conocimiento latente por cada examinado. En este paso se calcula el nivel de habilidad para cada examinado teniendo en cuenta las respuestas observadas y los

parámetros de los ítems. Este proceso concluye con la creación de un fichero que almacena los resultados de la estimación realizada en un HDFS.

2.3.2 Precondiciones

Para el correcto funcionamiento del algoritmo propuesto se debe tener en cuenta la siguiente precondición:

a) Los datos de entrada con vectores de respuestas dicotómicos por cada examinado sobre un grupo de ítems.

2.3.3 Entradas y salidas del algoritmo propuesto

La **entrada** del algoritmo es un fichero que contiene los vectores de respuestas dicotómicos de los examinados con respecto a un número de ítems. Cada vector de respuesta es una línea del fichero de entrada y cada elemento del vector se encuentra separado por coma.

Como **salida** se obtendrá un fichero que almacena los resultados de la estimación de conocimiento latente y el error estándar en la estimación por cada examinado. Cada resultado por examinado será una línea que contendrá la estimación como un par, donde cada elemento estará separado por el carácter coma.

2.4 Descripción formal del algoritmo propuesto

En el presente acápite se describe formalmente cada elemento del algoritmo propuesto.

2.4.1 Paso 1 Creación de los datasets distribuidos con vectores de respuestas e ítems con parámetros desconocidos.

Para la ejecución de tareas que permitan el procesamiento distribuido de los datasets, Spark define una arquitectura de alto nivel conformada por cinco elementos fundamentales, los cuales se explican a continuación (Guller, 2015):

- **Workers.** Son los nodos del clúster que proveen recursos de hardware para ejecutar un programa Spark. Estos recursos son CPU, memoria y almacenamiento. Cada worker ejecuta la aplicación de Spark como un proceso distribuido.
- **Cluster manager (Spark master).** Se encarga de proveer un planificador de alto nivel para administrar los recursos de los nodos workers. Estos recursos se administran en función de la necesidad de recursos de cómputo que tenga la tarea que se está ejecutando. Spark soporta tres tipos de Cluster manager: Standalone, YARN y Mesos.

- **Driver program.** Es una aplicación que usa Spark como una biblioteca. Provee el código para el procesamiento de datos que Spark ejecuta en los nodos workers. Este a su vez puede ejecutar varias tareas en un clúster.
- **Executors.** Es un proceso de la Máquina Virtual de Java (JVM, por sus siglas en inglés) que Spark crea en cada nodo worker para una aplicación. Este ejecuta el código de la aplicación concurrentemente en varios hilos. Cuando la aplicación termina de ejecutarse se cierran todos los ejecutores iniciados.
- **Task.** Es una pequeña unidad de trabajo que es enviada a un ejecutor. La misma es realizada en un hilo por un ejecutor en un nodo worker. Estas tareas son creadas por particiones de datos y un ejecutor las efectúa de forma concurrente.

A continuación se muestra la arquitectura de alto nivel de Spark, donde se ilustra la relación de cada uno de los elementos que la constituyen:

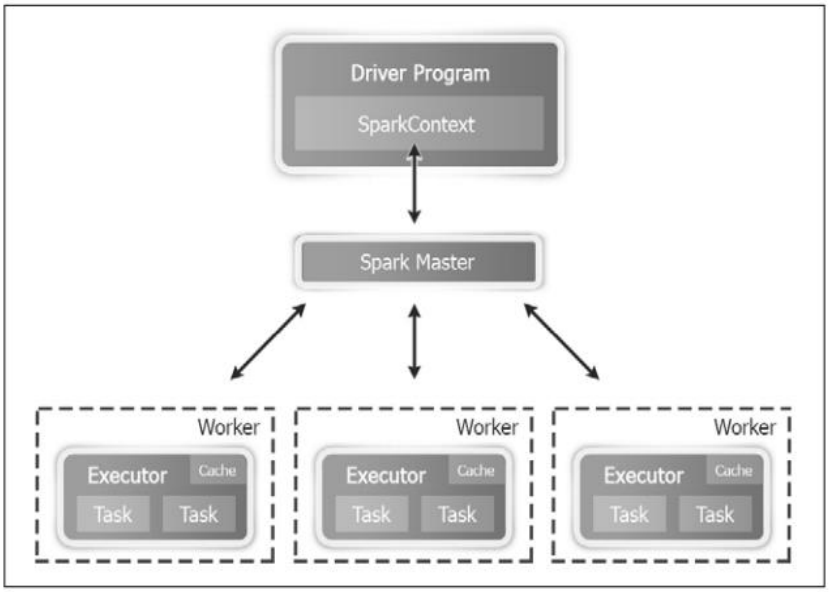


Figura 6. Arquitectura de alto nivel de Apache Spark (Yadav, 2015)

Con esto se asegura la ejecución concurrente de cada uno de los dataset distribuidos que sean creados a partir del procesamiento de los datos de entrada. Para crear un dataset distribuido se debe aplicar una transformación a partir de los datos de entrada y con estos obtener un RDD de algún objeto determinado. Un RDD de Spark es un dataset distribuido tolerante a fallos que trabaja como una colección de registros de cualquier tipo, que pueden ser objetos. Este RDD puede ser particionado y enviado a cada uno de los nodos ejecutores para ser procesado de forma paralela (Pentreath, 2015).

Variables compartidas en Apache Spark

Apache Spark permite a los desarrolladores hacer uso de dos tipos de variables compartidas que posibilitan la comunicación entre el máster/driver program y los nodos esclavos/workers. Están diseñadas para tener buen rendimiento cuando se procesan datos de forma distribuida y se requieren de variables para ser leídas o actualizadas (Chowdhury, Zaharia & Stoica, 2014).

Las variables compartidas con las que cuenta Apache Spark son las siguientes (Chowdhury, et al., 2014):

- **Broadcast.** Permite almacenar una gran pieza de datos destinada al modo de acceso de solo lectura para múltiples procesamientos en paralelo. Permite que los esclavos puedan acceder a datos cuando ejecutan operaciones distribuidas dentro de una transformación o acción. El Broadcast asegura que las variables se copien una sola vez en los nodos esclavos.
- **Accumulators.** Están disponibles a los nodos trabajadores para realizar operaciones de escritura de datos, donde estas operaciones tengan propiedad asociativa, por ejemplo: sumar valores, buscar mínimos y máximos dentro de un conjunto o contar elementos. Son variables tolerantes a fallos y su valor no puede ser accedido desde los nodos esclavos, solo se pueden leer desde el nodo máster.

Estas variables compartidas se aplican en el desarrollo de la propuesta de solución para el proceso de estimación de parámetros de un ítem y la estimación de conocimiento latente por cada examinado. Una vez que se creen los datasets distribuidos con vectores de respuestas dicotómicos e ítems con parámetros sin estimar, se hace necesario que los nodos esclavos tengan acceso al conjunto de valores de una función de distribución, para efectuar las operaciones concurrentes sobre las porciones de datos durante la estimación. Por otra parte, se necesita que ellos actualicen los resultados de sus cálculos en el nodo máster, por lo que se requiere de una variable acumuladora para este fin.

Para crear el dataset distribuido con los vectores de respuestas se aplicó una transformación a partir de la lectura del fichero con los datos de entrada. Durante el procesamiento, a cada registro se le eliminó el carácter coma que separaba los elementos del vector de respuesta dicotómico y se calculó el número de veces que se repite (frecuencia) el vector de respuesta en los datos de entrada. Con esto se reduce el tamaño de los datos a procesar. El vector de respuesta y su frecuencia fueron encapsulados en un objeto y se creó un RDD con dicho vector.

A continuación se muestra un fragmento de código que permite obtener un RDD de vectores de respuestas aplicando la API de Spark disponible para Java:

Tabla 5. Implementación del método para obtener un RDD de vectores de respuestas.

```
public JavaRDD<ItemResponseVector> loadItemResponsesVectors(String urlDataset,
JavaSparkContext jsc) {
      /*
            Permite cargar un dataset de una URL
            y por cada línea eliminar el caracter coma.
      */
        JavaRDD<String> linesVector = jsc.textFile(urlDataset);
        JavaRDD<String> linesWithoutColon = linesVector.map((String line) -> {
            return line.replaceAll(",", "");
        });
      /*
            Permite obtener un par distribuido formado por clave valor <K,V>,
            donde a cada vector de respuestas (K) se le asigna el valor 1 (V).
            Esto constituye una transformación distribuida sobre dataset.
      */
        JavaPairRDD<String, Integer> pairLineInt =
      linesWithoutColon.mapToPair((String line) -> {
            return new Tuple2<>(line, 1);
        });
      /*
            Calcula el número de apariciones o frecuencia
            que tiene un vector de respuestas en el dataset,
            De ellos se obtiene un par <K,V> con vectores de respuestas únicos.
            Esto constituye una acción distribuida sobre dataset.
      */
        JavaPairRDD<String, Integer> freqTable =
      pairLineInt.reduceByKey((Integer a, Integer b) -> {
            return a + b;
        }).distinct();
      /*
            Se transforma el resultado a un RDD de vectores de respuestas.
            La clase ItemResponseVector contiene el vector de respuestas y la
            frecuencia.
      */
        JavaRDD<ItemResponseVector> salida = freqTable.map((Tuple2<String,
      Integer> freqRow) -> {
            String responseString = freqRow._1;
            byte[] rv = new byte[responseString.length()];
            int i = 0;
            for (char letter : responseString.toCharArray()) {
                rv[i++] = Byte.parseByte("" + letter);
            }
            return new ItemResponseVector(rv, freqRow._2);
        });
        return salida;
    }
```

Un procedimiento similar se sigue para obtener el RDD de ítems con parámetros desconocidos. Las clases empleadas como objetos del RDD pertenecen a la librería Psychometrics, tanto para el vector de respuestas (***ItemResponseVector***) y los ítems con parámetros desconocidos (***Irm3PL***). Con estas salidas se procede a realizar la estimación de parámetros para los ítems mediante MML combinado con EM.

2.4.2 Paso 2 Estimación de parámetros de los ítems mediante MML combinado con EM

Para realizar el proceso de estimación de parámetros de un ítem, para los diferentes modelos logísticos de uno a tres parámetros, se aplica una adaptación del algoritmo EM descrito en el acápite 1.3.2 con el objetivo de lograr un mejor rendimiento al aplicar el estimador mencionado sobre un número considerable de ítems (Sehgal & Garg, 2014). Este algoritmo recibe como entrada una matriz de respuestas observadas con valores dicotómicos, donde las filas corresponden a cada i-ésimo examinado. Por cada iteración es elegido el máximo valor esperado de los datos de la función de verosimilitud de logaritmo natural con respecto a la distribución elegida. La distribución puede ser normal, condicional o continua (Hsu, 2000).

Para el procesamiento de los datos, la matriz de entrada se divide en bloques por cada una de las filas y estas se procesan en paralelo por cada ejecutor del clúster. El resultado de aplicar las ecuaciones 10 y 11 por cada fila es actualizado en el nodo maestro. Luego se aplica el paso M del algoritmo donde se estiman los parámetros de los ítems en paralelo a partir del resultado obtenido en el paso E. Se actualizan los criterios de parada en el nodo maestro al finalizar cada iteración.

Algoritmo 3. Paralelización del algoritmo EM para estimar parámetros en IRT

Entrada	• Sea Y matriz de tamaño $N\ x\ J$, donde N representa el número de examinados y J el número de ítems. • Sea $I' = (\delta'_{1,}\delta'_{2}, \dots, \delta'_{J})$ vector de ítems con parámetros desconocidos, donde $\delta'_{i} = \{t_1, t_2, \dots, t_T\}$ representa el conjunto de T parámetros de un ítem i. • Sea μ máximo cambio • Sea K tamaño distribución de probabilidades. • Sea Ω cantidad máxima de iteraciones.
Salida	• Vector de ítems con parámetros estimados $I = (\delta_{1,}\delta_{2}, \dots, \delta_{J})$
Pasos: • Inicializar D variable de difusión que almacena los vectores $Q = (q_1, q_2, \dots, q_K)$ y $\pi = (\pi_1, \pi_2, \dots, \pi_K)$. • Crear dataset distribuido a partir I'. • Inicializar $s = 0$, $e = 1 +$ μ	

- Inicializar variable acumuladora A de máximo valor
- Inicializar variable acumuladora B que almacena $n_k^{(s)}$ y $r_{jk}^{(s)}$
- Mientras $s < \Omega$ y $e > \mu$
    a. Ejecutar Paso E. Ver Algoritmo 4.
    b. Ejecutar Paso M. Ver Algoritmo 5.
    c. Recibir de cada ejecutor el valor $m = |\delta_1^{(s)} - \delta_1^{(s-1)}|$
    c. Actualizar en A lo obtenido en m
    d. $e = valor(A)$
    e. $s = s + 1$
    f. Reiniciar B

En este algoritmo se ejecuta el paso E donde se actualiza la variable acumuladora B por cada ejecutor del clúster, aplicando las ecuaciones 10 y 11 en cada una de las filas de Y. La variable de difusión D almacena los valores de distribución de probabilidades que representan la variable latente θ_i por cada estudiante i. Luego se procede a ejecutar el paso M donde se actualizan los parámetros de los ítems en el vector I que es almacenado en un RDD.

Algoritmo 4. Paralelización del paso E para la obtención de las estadísticas suficientes y completas

Entrada	• Sea Y matriz de tamaño $N\ x\ J$, donde N representa el número de examinados y J el número de ítems. • Sea $I' = (\delta'_1, \delta'_2, \dots, \delta'_J)$ vector de ítems con parámetros desconocidos, donde $\delta'_i = \{t_1, t_2, \dots, t_T\}$ representa el conjunto de T parámetros de un ítem i. • D variable de difusión que almacena los vectores $Q = (q_1, q_2, \dots, q_K)$ y $\pi = (\pi_1, \pi_2, \dots, \pi_K)$. • Sea B variable acumuladora que almacena $n_k^{(s)}$ y $r_{jk}^{(s)}$ • Sea K tamaño distribución de probabilidades.
Salida	• Salida vacía
Pasos: 1. Por cada ejecutor recibir $y_i \epsilon Y$ y hacer:	

a. $n_k^{(s)} = \sum_{i=1}^{N} \frac{\pi_k^{(s)} \prod_{j=1}^{J} P(q_k|\delta_j^{(s)})^{y_{ij}} [1-P(q_k|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}{\sum_{k'=1}^{K} \pi_{k'}^{(s)} \prod_j^J P(q_{k'}|\delta_j^{(s)})^{y_{ij}} [1-P(q_{k'}|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}$

b. $r_{jk}^{(s)} = \sum_{i=1}^{N} \frac{y_{ij}\pi_k^{(s)} \prod_{j=1}^{J} P(q_k|\delta_j^{(s)})^{y_{ij}} [1-P(q_k|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}{\sum_{k'=1}^{K} \pi_{k'}^{(s)} \prod_j^J P(q_{k'}|\delta_j^{(s)})^{y_{ij}} [1-P(q_{k'}|\delta_j^{(s)})^{y_{ij}}]^{1-y_{ij}}}$

c. Actualizar en el máster la variable acumuladora B con los valores calculados en los pasos a y b.

El objetivo del paso E para el proceso de estimación es actualizar en el nodo máster la variable acumuladora B que contiene las estadísticas $n_k^{(s)}$ y $r_{jk}^{(s)}$. Esta variable sirve como entrada al algoritmo que se ejecuta en el paso M. Para ello se guardada la información calculada en una variable de difusión para que estos valores puedan ser utilizados por los ejecutores en el proceso de actualización de los parámetros de los ítems en I'.

Algoritmo 5. Paralelización del paso M para la estimación de parámetros.

Entrada	• Sea $I' = (\delta'_{1,}\delta'_2, \dots, \delta'_J)$ vector de ítems con parámetros desconocidos, donde $\delta'_i = \{t_1, t_2, \dots, t_T\}$ representa el conjunto de T parámetros de un ítem i. • Sea D variable de difusión que almacena los vectores de la distribución de probabilidades $Q = (q_1, q_2, \dots, q_K)$ y $\pi = (\pi_1, \pi_2, \dots, \pi_K)$. • Sea B variable acumuladora que almacena $n_k^{(s)}$ y $r_{jk}^{(s)}$
Salida	• Sea $I = (\delta_{1,}\delta_2, \dots, \delta_J)$ vector de ítems con parámetros estimados.

Pasos:

1. Inicializar variable de difusión H a partir de B
2. Por cada ejecutor recibir $\delta'_j \epsilon\, I$ y hacer:

   d. Resolver el sistema de ecuaciones $\sum_{k=1}^{K} \frac{r_{jk}^{(s)} - n_k^{(s)} P(q_k|\delta_j'^{(s)})}{[1-P(q_k|\delta_j'^{(s)})]P(q_k|\delta_j'^{(s)})} \frac{\partial P(q_k|\delta_j'^{(s)})}{\partial \delta_{tj}'^{(s)}} = 0$ para obtener el conjunto T de parámetros

   a. $\delta'_j = T$

En el paso M resulta de gran importancia el uso de un optimizador de mínimo sin restricciones (***DefaultUncminOptimizer***), este optimizador es una clase presente en la biblioteca Psychometrics para resolver el sistema de ecuaciones del paso 2.d. El mismo contiene métodos que permiten minimizar una curva suave de varias variables y se busca minimizar el valor del logaritmo natural para la función de verosimilitud de un determinado ítem (Schnabel, Koontz, & Weiss, 1982). Para cada ejecutor crea una función de verosimilitud en cada ítem j a partir de la variable acumuladora H, el conjunto δ'_j que contiene valores iniciales para cada parámetro y la variable de difusión D. Esta función de verosimilitud es recibida por el optimizador y con esta se resuelve el sistema buscando para cada parámetro qué valor tiene mayor probabilidad de ocurrir. La misma se encuentra de igual forma disponible en la mencionada biblioteca a través de la clase ***ItemLogLikelihood.***

Con los resultados obtenidos por el optimizador se obtiene un vector de valores propuestos para los parámetros de un ítem. Con este vector se actualizan los parámetros de los ítems y se calcula el máximo cambio, que indica en cuanto variaron los parámetros estimados, con respecto a los existentes.

2.4.3 Paso 3 Estimación de conocimiento latente por cada examinado mediante IRT

En este paso se reciben los RDD que contienen la matriz Y y el vector I obtenido en el paso 2. Con estos elementos se crea una variable de difusión a partir de los ítems con parámetros estimados para que el resto de los ejecutores puedan utilizarlo para estimar conocimiento por cada examinado. El resultado que se obtiene como salida es almacenado en un fichero de texto en el nodo máster.

Algoritmo 6. Paralelización del procedimiento para la estimación de conocimiento latente.

Entrada	• Sea Y matriz de tamaño $N\,x\,J$, donde N representa el número de examinados y J el número de ítems. • Sea $I = (\delta_{1,}\delta_2, \dots, \delta_J)$ vector de ítems con parámetros estimados. • Sea θ_i conocimiento latente del examinado i definido a priori. • Sea ϕ cantidad máxima de iteraciones. • Sea ξ el error estándar en la estimación. • Sea Δ el cambio mínimo en la estimación.
Salida	• Conjunto R formado por los pares $(\theta_i, SE(\theta_i))$.

Pasos:

1. Inicializar variable de difusión G a partir de I
2. Por cada ejecutor recibir $y_i \epsilon Y$ y hacer:
   a. Inicializar $s = 0$, $e = 0$
   b. Hacer
      i. $\theta^{s+1}{}_i = \theta^s{}_i + \left. \sum_{j=1}^{J} a_j \left[u_j - P_j(\theta^s{}_j)\right] \right/ \sum_{j=1}^{J} a_j{}^2 P_j(\theta^s{}_j) Q_j(\theta^s{}_j)$
      ii. $s = s + 1$
      iii. $e_i = \left. 1 \right/ \sqrt{\sum_{j=1}^{J} a_j{}^2 P_j(\theta^s{}_j) Q_j(\theta^s{}_j)}$

   Mientras $|\theta^s{}_i - \theta^{s-1}{}_i| > \Delta$ y $e > \xi$ y $s < \phi$

2.5 Conclusiones del capítulo

- El algoritmo propuesto permite estimar conocimiento latente para cada examinado a partir de las respuestas observadas para cada uno de los ítems evaluativos y almacenar el resultado de la estimación para su posterior análisis.
- El uso del algoritmo EM permitió aplicar un estimador basado en MML para estimar los parámetros de los ítems evaluativos a partir de las respuestas observadas, este proceso es fundamental para luego estimar el conocimiento latente por cada examinado.
- El análisis del algoritmo EM para el modelo IRT permitió identificar que se puede procesar la entrada de forma independiente por filas debido a que no existen dependencias entre ellas para la realización de los cálculos; lo que facilitó el diseño e implementación de una solución paralela.
- El uso del paradigma MapReduce sobre la herramienta Apache Spark permitió obtener una solución tolerante a fallos que permite el procesamiento de datos educacionales masivos para estimar conocimiento latente.

CAPÍTULO 3. VALIDACIÓN DE LA PROPUESTA

En este capítulo se presenta la aplicación de la adaptación del algoritmo IRT para estimar conocimiento latente sobre Datos Educacionales Masivos, a través de una capa de servicios REST. Se realiza la validación de la propuesta de solución mediante pruebas de rendimiento computacional que permiten medir Speedup y eficiencia. Para ello se describe el ambiente donde se realizaron las pruebas y se comparan los resultados de la estimación y el rendimiento de la propuesta con respecto al algoritmo original. Estas mediciones se realizan escalando el tamaño de la entrada y el número de procesadores.

3.1 Solución propuesta

La adaptación del algoritmo IRT para estimar conocimiento latente sobre Datos Educacionales Masivos está accesible a través de una API de servicios REST. Con esta se puede acceder a los servicios de estimación de parámetros de un ítem y estimación de conocimiento para los examinados mediante el envío de peticiones por el protocolo HTTP. Las respuestas del resultado del procesamiento efectuado por el clúster de computadoras son recibidas en formato JSON por el mismo protocolo. Este aporte práctico permitirá que otras plataforma educativas puedan consumir estos servicios disponibles y documentados para medir el conocimiento latente de los estudiantes en los cursos de forma global o por materias, conocer los parámetros de los ítems de sus instrumentos evaluativos en línea (discriminador, dificultad y probabilidad de ser contestados correctamente al azar) y predecir el éxito frente a un instrumento evaluativo.

A continuación se presenta la vista general de la capa de servicios REST de la propuesta de solución:

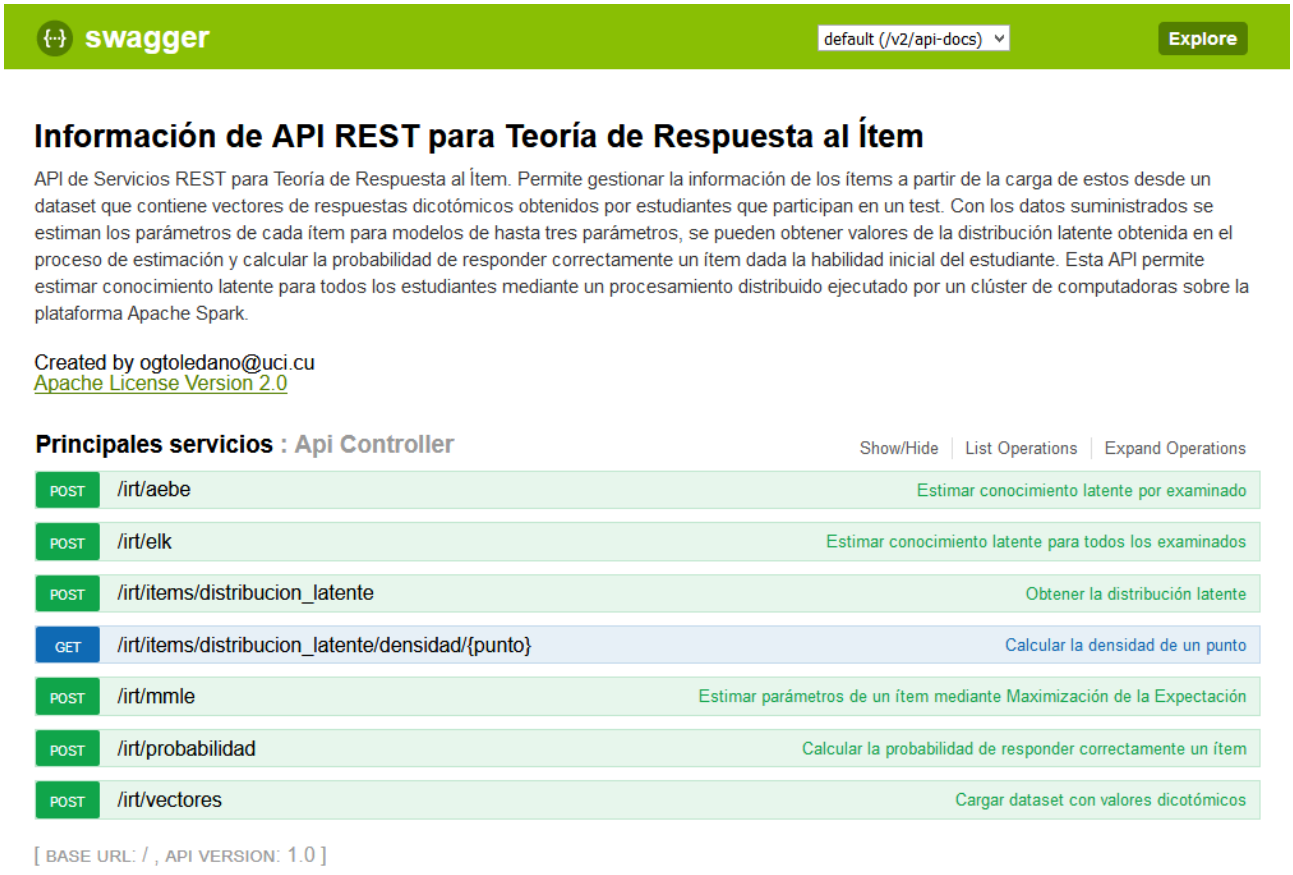
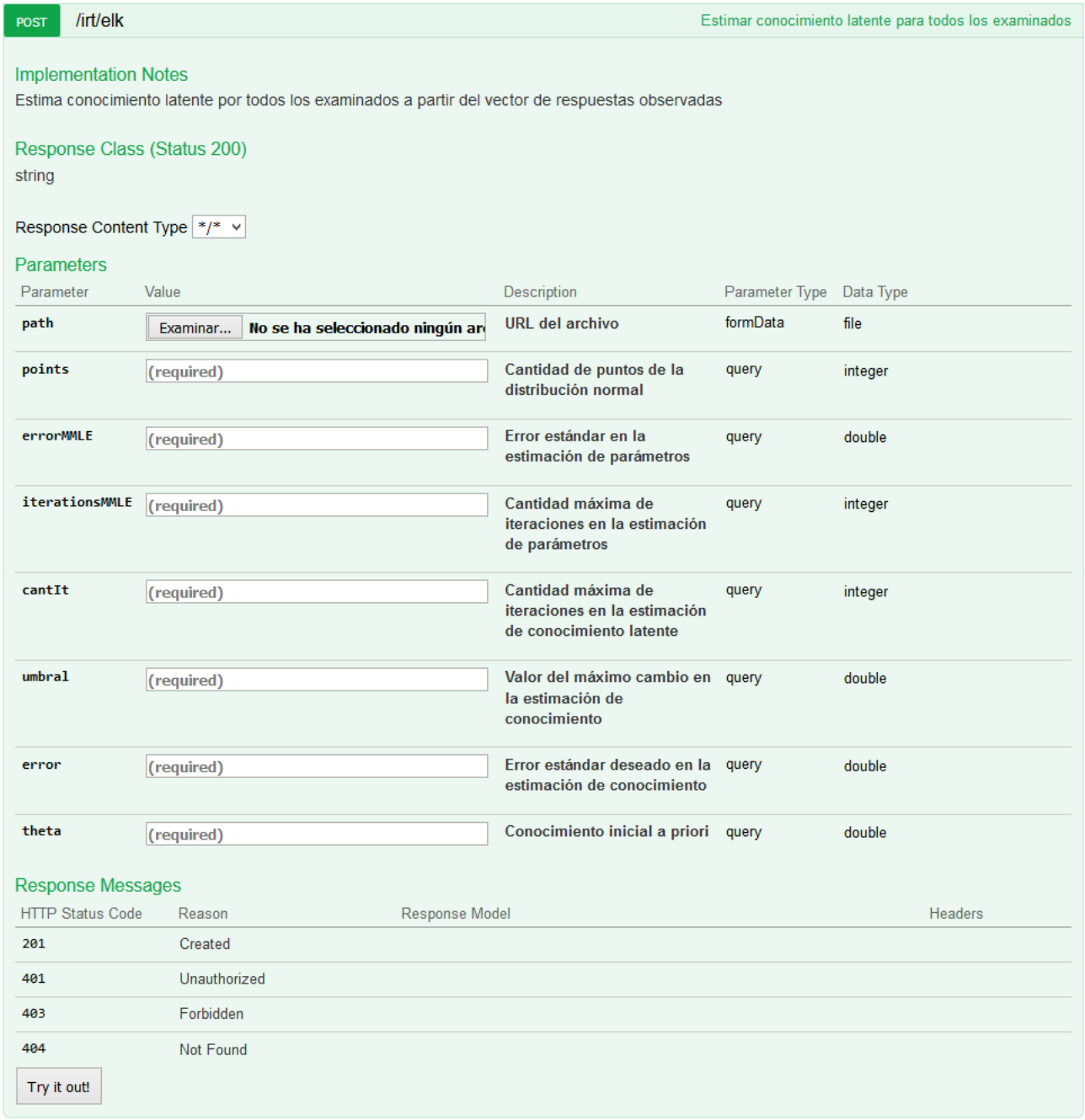


Figura 7. Vista general de la capa de servicios REST de la propuesta de solución

Para realizar el proceso de estimación de conocimiento latente se carga un dataset que contiene una entrada con los vectores de respuestas dicotómicos ofrecidos por los estudiantes examinados. Se especifican los criterios de parada del algoritmo y se envía la petición para efectuar la estimación de conocimiento latente para todos los examinados.

Este servicio de estimación está accesible a través de una URL. Una vez suministrados los datos de entrada necesarios se puede efectuar la petición y el servidor procede a ejecutar el procesamiento de forma distribuida sobre un clúster de computadoras. La siguiente figura muestra una vista del servicio que permite realizar lo anterior.



POST /irt/elk — Estimar conocimiento latente para todos los examinados

Implementation Notes

Estima conocimiento latente por todos los examinados a partir del vector de respuestas observadas

Response Class (Status 200)

string

Response Content Type */*

Parameters

Parameter	Value	Description	Parameter Type	Data Type
path	Examinar... No se ha seleccionado ningún ar	URL del archivo	formData	file
points	(required)	Cantidad de puntos de la distribución normal	query	integer
errorMMLE	(required)	Error estándar en la estimación de parámetros	query	double
iterationsMMLE	(required)	Cantidad máxima de iteraciones en la estimación de parámetros	query	integer
cantIt	(required)	Cantidad máxima de iteraciones en la estimación de conocimiento latente	query	integer
umbral	(required)	Valor del máximo cambio en la estimación de conocimiento	query	double
error	(required)	Error estándar deseado en la estimación de conocimiento	query	double
theta	(required)	Conocimiento inicial a priori	query	double

Response Messages

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

Try it out!

Figura 8. Servicio para estimar conocimiento latente para todos los examinados a partir de las respuestas observadas

Una vez ejecutada la petición al servidor se almacena el resultado en un fichero en el nodo máster cuya salida es el conjunto R formado por los pares $(\theta_i, SE(\theta_i))$. El servidor devuelve además el mismo resultado en formato JSON.

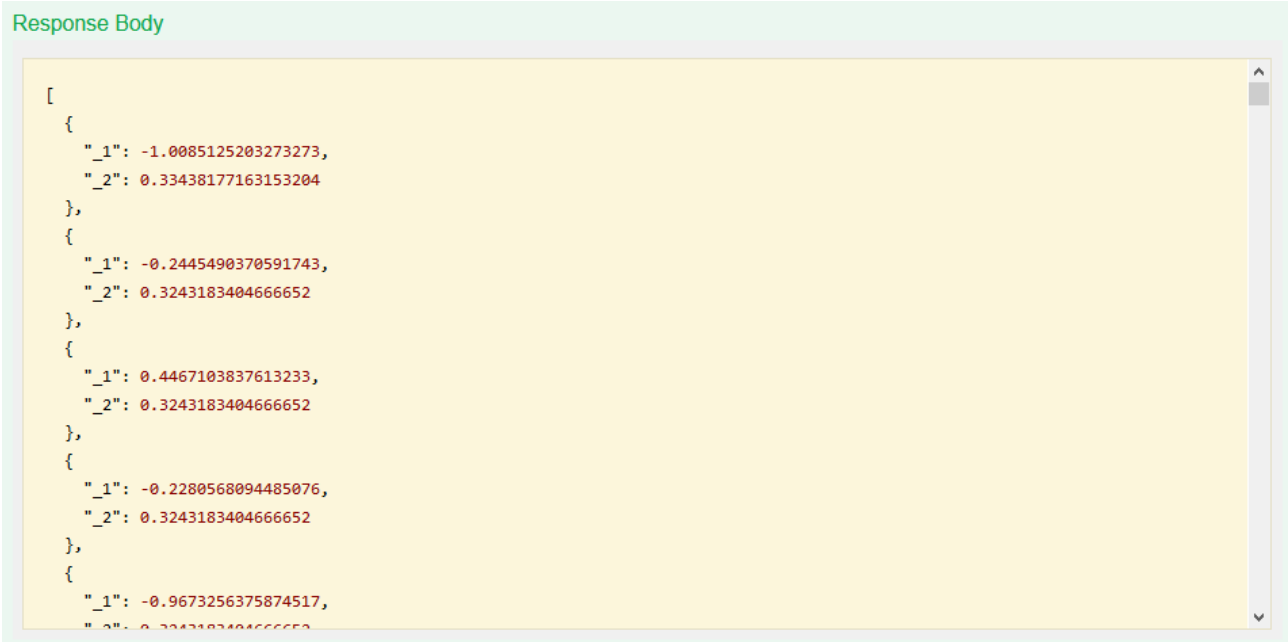


Figura 9. Respuesta enviada desde el servidor en formato JSON que representa el resultado de estimar conocimiento latente por cada examinado

3.2 Valoración del costo computacional de la propuesta de solución

Para realizar la medición del rendimiento computacional de la propuesta de solución se realizó un estudio de los marcos de trabajo existentes que permiten calcular métricas de rendimiento para algoritmos paralelizados. Estos marcos de trabajo o enfoques se le conocen como Ley de Amdalh y Ley de Gustafson. Estos proveen modelos matemáticos que permiten calcular el incremento de velocidad de los sistemas de información, donde una parte de estos puede ser mejorada o paralelizable, para poder ser ejecutada por más de una unidad de procesamiento de manera simultánea (Nielsen, 2016).

3.2.1 Ley de Amdalh

Este marco de trabajo permite caracterizar el impacto del paralelismo de datos para la mejora del rendimiento, cuando el tamaño del problema o la entrada de datos son constantes. Se define que un algoritmo está conformado por dos partes, una porción de código que es paralelizable denotada por α_{par} y otra porción que intrínsecamente no se puede paralelizar que se denotaría por α_{seq}. Con esto se tiene que $\alpha_{par} + \alpha_{seq} = 1$, que representa el total del código (Nielsen, 2016).

El tiempo que demora en ejecutarse un algoritmo paralelo con P procesadores sería el siguiente:

$$\boldsymbol{tp} = \frac{\boldsymbol{\alpha_{par} t_1}}{\boldsymbol{P}} + \boldsymbol{\alpha_{seq} t_1} \quad [14]$$

Donde t_1 representa el tiempo que demora en ejecutarse el algoritmo secuencialmente.

Por tanto el incremento de velocidad o Speedup para P procesadores se puede calcular como:

$$\boldsymbol{S(P) = t_1/t_p} \quad [15]$$

Donde t_p representa el tiempo de ejecución del algoritmo paralelo con P procesadores.

A partir de lo anterior la Ley de Amdalh plantea que el máximo incremento de velocidad que puede alcanzar un sistema de información está condicionado por la porción del código que no puede ser paralelizable. Por lo que el límite del $S(P)$ cuando P tiende a infinito nunca sobrepasará a la razón $1/\alpha_{seq}$.

$$\boldsymbol{\lim_{p\to\infty} S(P) = 1/\alpha_{seq}} \quad [16]$$

El área del Big Data no está exenta de este fenómeno, donde las partes no paralelizables del algoritmo hacen un cuello de botella "bottleneck" que influye en la ganancia de Speedup cuando se incrementa el número de nodos/procesadores en la solución. Lo que demuestra una baja eficiencia en las soluciones por lo que llegan a un límite de menor tiempo cuando alcanzan un determinado número de procesadores (Richins, Ahmed, Clapp, & Reddi, 2018).

3.2.2 Ley de Gustafson

Este enfoque provee de un nuevo punto de vista para calcular de forma escalada el Speedup. En este marco nunca se asume que el tamaño de los datos es constante, pues el tamaño varía en dependencia del número de procesadores disponibles para la ejecución de la solución. En la medida que aumentan los datos se incrementan el número de procesadores para mantener un tiempo de ejecución constante (Nielsen, 2016).

A partir de esto el Speedup se calcula como sigue (Nielsen, 2016):

$$\boldsymbol{S(P) = \alpha_{seq} + P\ \alpha_{par}} \quad [17]$$

De esta forma cuando crece el tamaño del problema para un α_{seq} constante si aumenta el número de procesadores aumenta la ganancia de velocidad $S(P)$. La siguiente figura ilustra lo anterior:

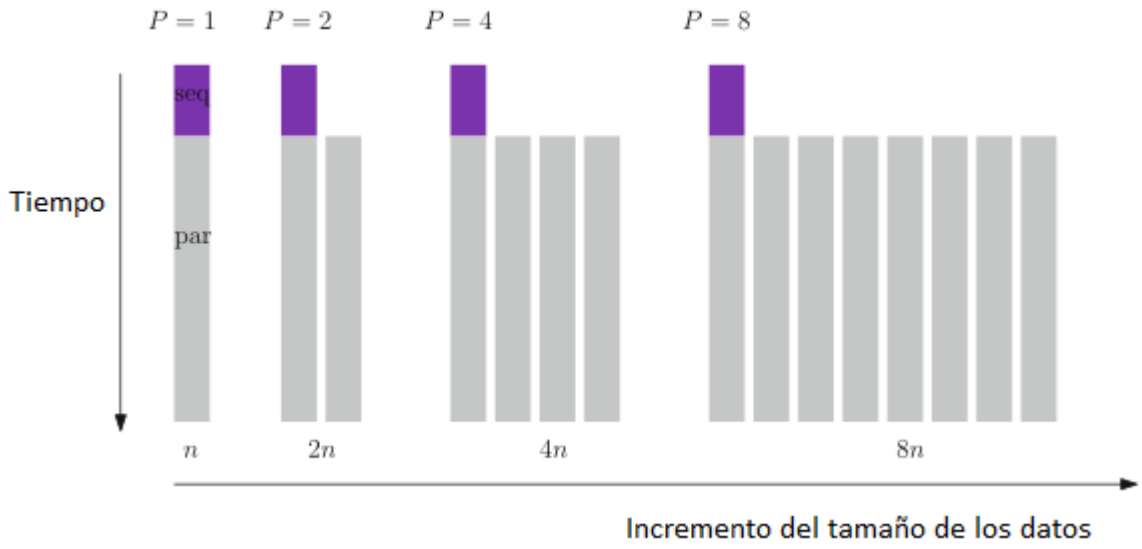


Figura 10. Comportamiento constante del tiempo de ejecución si aumentan al mismo tiempo el tamaño de los datos y los procesadores (Nielsen, 2016)

A partir de que las mediciones se realizarán tomando datos de tamaño fijo se aplicará el enfoque que brinda la ley de Amdhal. Este permitirá medir el Speedup conociendo los tiempos de ejecución a posteriori del algoritmo secuencial y su solución paralelizada. Con el Speedup obtenido se procederá a calcular la eficiencia de la solución y qué tanto mejora el tiempo de esta en la medida que aumentan los procesadores.

3.2.3 Pruebas de rendimiento computacional

Para la realización de las pruebas de rendimiento computacional se desplegó la propuesta de solución en un clúster de computadoras usando la herramienta Apache Spark en su versión 2.1.1. Se aplicó el modo Standalone de despliegue, donde se utilizan las funciones nativas de la herramienta. El clúster estaba conformado por una estación de trabajo que actuó como máster y cinco estaciones de trabajo utilizadas como nodos trabajadores. Se debe tener en cuenta que las pruebas fueron realizadas en un entorno no dedicado de nodos que pertenecen a la misma subred.

A continuación se muestran las especificaciones de hardware del clúster de computadoras empleado en las pruebas:

Tabla 6. Especificaciones de hardware del clúster de computadoras empleado en las pruebas

Estación de trabajo Máster		
Tipo de procesador	**Cantidad de núcleos**	**Memoria principal**

Intel Celeron CPU G1839 2.8GHz	2	4GB DDR3
Estaciones de trabajo usadas como nodos trabajadores		
Tipo de procesador	**Cantidad de núcleos**	**Memoria principal**
Intel Core i3-4160 3.6GHz	4	4GB DDR3
Intel Celeron CPU G1839 2.8GHz	2	4GB DDR3
Intel Core i3-4160 3.6GHz	4	4GB DDR3
Intel Celeron CPU G1839 2.8GHz	2	4GB DDR3
Intel Core i3-4160 3.6GHz	4	4GB DDR3
Características de la red de datos		
Red cableada Fast Ethernet 100 Mbps		

Como especificación de software y sistema operativo instalado en las estaciones de trabajo empleadas en el clúster están las siguientes:

Tabla 7. Especificaciones de software y sistema operativo

Sistema operativo
Ubuntu 16.04 LTS 64 bits, versión del núcleo: 4.4.0-121-generic
Software necesarios instalados
Máquina Virtual de Java OpenJDK 8 64bits, Versión de Java: 1.8.0_162
Scala 2.11.8
Apache Spark 2.1.1, versión de Hadoop 2.7.3

Para la ejecución de las pruebas se utilizaron datasets formados una matriz Y de tamaño $N\ x\ J$. Se escalaron los datos de pruebas aumentando el número de estudiantes N y los ítems J. Se realizaron las pruebas en 10 ambientes distintos del clúster Spark incrementando el número de ejecutores/procesadores en cada caso; combinando para un igual número de procesadores velocidades de reloj diferentes al ser los nodos esclavos, computadoras de diferente fabricante.

Se determinaron los siguientes requisitos para todas las mediciones realizadas:

- Cantidad máxima de iteraciones para la estimación de parámetros: $\Omega = 100$
- Cantidad máxima de iteraciones para la estimación de conocimiento: $\phi = 100$

La intensión de estas restricciones es realizar un igual número de corridas de los pasos lógicos 2 y 3 de la propuesta de solución, para ello no se tuvieron en cuenta el resto de los criterios de parada definidos en ambos pasos.

Los juegos de datos empleados fueron seis datasets con matrices de tamaño [2000x50], [2000x200], [5000x50], [5000x200], [10000x50] y [10000x200] en 10 ambientes distintos de configuración escalando el número de ejecutores de 4 a 16 núcleos. Se realizaron 20 corridas del de la propuesta de solución por cada uno de los datasets y se tomaron los tiempos más representativos de cada muestra. Por cada ejecutor existente en cada experimento se asignó 1GB de memoria principal. Ver Anexo 1: Resultados de las pruebas de rendimiento realizadas.

Para elegir los valores más representativos de cada muestra en los 10 ambientes de configuración se realizaron pruebas de normalidad usando la técnica Shapiro Wilk. Esta permite realizar pruebas de normalidad a una determinada muestra cuyo tamaño es inferior a 30. Se considera muy potente en comparación con otras técnicas de su tipo y su fundamento estadístico está basado en una gráfica de probabilidad en la que se considera la regresión de las observaciones sobre los valores esperados de la distribución hipotetizada (Pedrosa, Juarros-Basterretxea, Robles-Fernández, Basteiro, & García-Cueto, 2015). De las 60 muestras analizadas el 38,33 % seguían una distribución normal con un intervalo de confianza de un 95 %. Para las muestras que tenían una distribución normal, fue elegido como valor más representativo la media de tiempo y para el resto de las muestras que no seguían una distribución normal la mediana.

Se realizaron pruebas para medir el tiempo de ejecución de la variante secuencial del algoritmo manteniéndose los mismos parámetros aplicados como condición de parada y los mismos juegos de datos. La medición se realizó ejecutando los mismos pasos de la propuesta de solución. Para la medición se utilizó una estación de trabajo con las mismas prestaciones del más potente de los nodos trabajadores.

Tabla 8. Especificaciones de hardware para ejecutar el algoritmo secuencial

Estación de trabajo usada para ejecutar el algoritmo secuencial		
Tipo de procesador	**Cantidad de núcleos**	**Memoria principal**
Intel Core i3-4160 3.6GHz	4	4GB DDR3

A continuación se muestran los resultados de las mediciones realizadas para datasets dicotómicos de diferentes tamaños escalando el número de procesadores de dos hasta ocho. En cada caso se utilizó 1GB de RAM por cada núcleo de procesamiento. Para describir las prestaciones de los ambientes se utilizó la nomenclatura CxP+CxP (definida por el autor). En esta C indica la cantidad

de computadoras de un tipo que tienen P procesadores. Cuando solo se emplean 2 computadoras con 4 procesadores cada una, la nomenclatura sería 2x4. Con esto se indica además la cantidad total de núcleos en el clúster, que para el ejemplo son 8. La unidad de medida para las muestras de tiempo en las mediciones es milisegundos (ms).

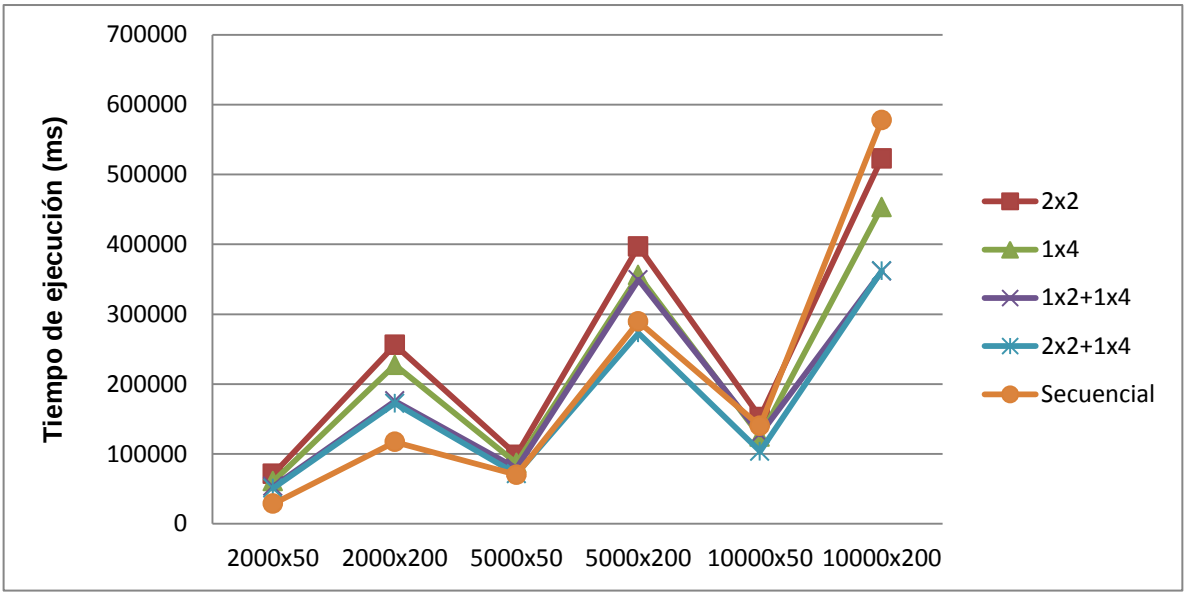


Figura 11. Gráfico que muestra las comparaciones entre los tiempos de ejecución del algoritmo secuencial y ambientes del clúster hasta ocho núcleos

En la gráfica se puede observar que para la entrada de datos 10000x200 se comienza a notar una mejora en el rendimiento a partir del ambiente 2x2. Por lo que para los casos donde la entrada de datos es pequeña se hace más factible aplicar la variante secuencial del algoritmo. Se debe tener en cuenta que la operación de entrada y salida a través de la red mediante el intercambio de datos entre el nodo máster y los esclavos penaliza el rendimiento.

Las siguientes mediciones muestran los resultados con ambientes de configuración más potentes:

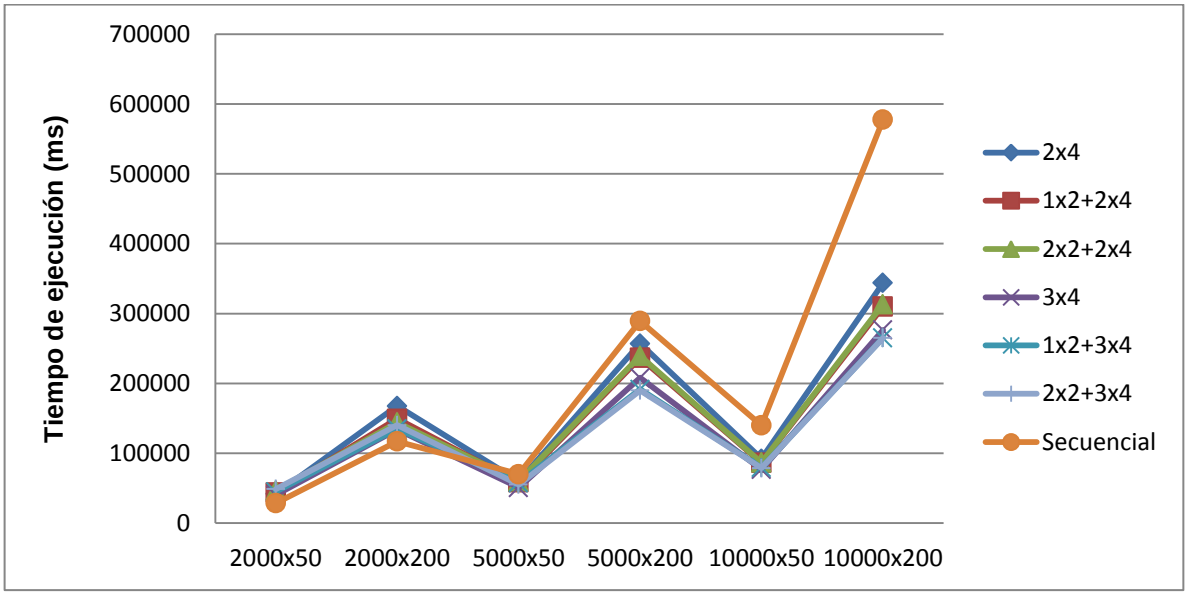


Figura 12. Gráfico que muestra las comparaciones entre los tiempos de ejecución del algoritmo secuencial y ambientes del clúster hasta 16 núcleos

En estos ambientes de configuración se puede observar que a partir del tamaño de la entrada 5000x200 en adelante se obtienen mejores tiempos que el algoritmo secuencial. Con los resultados anteriores se proceden a calcular las métricas de rendimiento para la entrada de datos a partir de las cuales existe mayor ganancia de velocidad.

Tabla 9. Métricas de rendimiento para cada ambiente con ganancia de velocidad

Ambiente: 2x4		
Tamaño del dataset [KxN]	**Speedup** $S(P) = t_s/t_p$	**Eficiencia** $E(p) = S(P)/P$
5000x200	1.2557	0.1569
10000x50	1.5215	0.1901
10000x200	1.6795	0.2099
Ambiente: 1x2+2x4		
Tamaño del dataset [KxN]	**Speedup** $S(P) = t_s/t_p$	**Eficiencia** $E(p) = S(P)/P$
5000x200	1.2230	0.1223
10000x50	1.6194	0.1619
10000x200	1.8632	0.1863

Ambiente: 2x2+2x4		
Tamaño del dataset [NxJ]	**Speedup** $S(P) = t_s/t_p$	**Eficiencia** $E(p) = S(P)/P$
5000x200	1.2086	0.1007
10000x50	1.6126	0.1343
10000x200	1.8426	0.1535
Ambiente: 3x4		
Tamaño del dataset [NxJ]	**Speedup** $\mathbf{S(P) = t_s/t_p}$	**Eficiencia** $\mathbf{E(p) = S(P)/P}$
5000x200	1.3918	0.1159
10000x50	1.8267	0.1522
10000x200	2.0884	0.1740
Ambiente: 1x2+3x4		
Tamaño del dataset [NxJ]	**Speedup** $S(P) = t_s/t_p$	**Eficiencia** $E(p) = S(P)/P$
5000x200	1.5114	0.1079
10000x50	1.7599	0.1257
10000x200	2.1769	0.1554
Ambiente: 2x2+3x4		
Tamaño del dataset [NxJ]	**Speedup** $S(P) = t_s/t_p$	**Eficiencia** $E(p) = S(P)/P$
5000x200	1.5240	0.0952
10000x50	1.7745	0.1109
10000x200	2.1780	0.1361

Para analizar si existe equilibrio en el procesamiento en paralelo se realizaron otros experimentos para conocer la cantidad de tareas ejecutadas por cada nodo trabajador y el número de bloques procesados por cada uno. Para ello se eligieron tres ambientes de configuración y se seleccionaron datasets con tamaño 10000x50 y 10000x200.

Tabla 10. Distribución de tareas y bloques por nodo trabajador

Ambiente: 2x2+2x4		**Dataset: [10000x50]**		**Dataset: [10000x200]**	
Nodo trabajador	**Núcleos**	**Tareas**	**Bloques**	**Tareas**	**Bloques**
Intel Core i3-4160 3.6GHz	4	4767	70	5270	89
Intel Core i3-4160 3.6GHz	4	5171	73	4668	86
Intel Celeron CPU G1839 2.8GHz	2	605	39	404	54
Intel Celeron CPU G1839 2.8GHz	2	404	38	605	55
Ambiente: 3x4		**Dataset: [10000x50]**		**Dataset: [10000x200]**	
Nodo trabajador	**Núcleos**	**Tareas**	**Bloques**	**Tareas**	**Bloques**
Intel Core i3-4160 3.6GHz	4	4671	84	1009	55
Intel Core i3-4160 3.6GHz	4	1009	57	4764	83
Intel Core i3-4160 3.6GHz	4	5267	89	5174	86
Ambiente: 1x2+3x4		**Dataset: [10000x50]**		**Dataset: [10000x200]**	
Nodo trabajador	**Núcleos**	**Tareas**	**Bloques**	**Tareas**	**Bloques**
Intel Core i3-4160 3.6GHz	4	5675	82	7908	110
Intel Core i3-4160 3.6GHz	4	808	47	808	59
Intel Core i3-4160 3.6GHz	4	5683	83	1009	60
Intel Celeron CPU G1839 2.8GHz	2	605	46	3046	77

Analizando los resultados, se puede observar que cada dataset utilizado genera un número determinado de trabajos que es igual para cada ambiente de configuración. Estos trabajos se dividen en fases/stages que constituyen una colección de tareas/tasks que se realizan en paralelo por cada ejecutor. Se hace notable que la cantidad de bloques procesados por cada nodo trabajador no está equilibrada y por tanto lo mismo ocurre con la cantidad de tareas procesadas por cada uno. Esto sucede debido a que el despliegue de la propuesta de solución es un clúster en modo Standalone que sigue una distribución de tareas que se realiza siguiendo una estrategia FIFO (Primero en Entrar Primero en Salir, por sus siglas en inglés) por tanto las tareas son asignadas en el orden en que un recurso está disponible. A esto se le une que el entorno escogido para las pruebas es un ambiente no dedicado, con nodos que poseen diferentes prestaciones de hardware y una infraestructura de red que por su uso generaba en gran medida fluctuaciones durante el intercambio de datos entre los esclavos y el máster. Este intercambio a

través de la red es una operación costosa de entrada y salida que constituye un principal cuello de botellas en la ganancia de velocidad.

3.3 Conclusiones del capítulo

- Con el desarrollo de la propuesta se obtiene una adaptación del algoritmo IRT para estimar conocimiento latente sobre Datos Educacionales Masivos mediante un clúster Spark desplegado en modo Standalone. Dicha solución puede ser accesible a través de una API de servicios REST.
- El uso de la técnica de Shapiro Wilk permitió realizar pruebas de normalidad a las diferentes muestras que se obtenían con la realización del experimento en cada ambiente de pruebas; con esto se hizo posible elegir las mediciones de tiempo más representativas de cada muestra observada.
- La aplicación de pruebas de rendimiento computacional permitieron evaluar la propuesta de solución, demostrando que para entradas donde los datos comienzan a crecer, esta tiene un mejor rendimiento que el algoritmo secuencial y se logra ganancia de velocidad.
- El uso de un ambiente no dedicado para el despliegue de la propuesta de solución hace que exista poco equilibrio en la distribución de tareas y bloques por lo que se hace necesario, para ganar un mejor rendimiento, aplicar otras formas de planificación de clúster y despliegue para Spark.

CONCLUSIONES

- El estudio del modelo basado en Teoría de Respuesta al Ítem permitió caracterizar el proceso de estimación de conocimiento latente e identificar métodos para la estimación de parámetros a partir de las respuestas observadas, posibilitando elegir el estimador MML combinado con EM, al tener buen rendimiento cuando los datos crecen considerablemente.
- El análisis de varios modelos de programación distribuida permitió identificar a MapReduce como un modelo adecuado para el desarrollo de la propuesta de solución al ser altamente tolerante a fallos.
- El análisis del algoritmo EM y el modelo de estimación de habilidad por examinados que ofrece IRT evidenció que se puede procesar la entrada de forma independiente por filas; lo que facilitó el diseño e implementación de una solución paralela.
- Con el desarrollo de la propuesta se obtiene una adaptación del algoritmo IRT para estimar conocimiento latente sobre Datos Educacionales Masivos, mediante un clúster Spark desplegado en modo Standalone y accesible a través de una API de servicios REST.
- La aplicación de pruebas de rendimiento computacional permitieron evaluar la propuesta de solución, demostrando que para entradas donde los datos comienzan a crecer esta tiene un mejor rendimiento que el algoritmo secuencial y se logra una ganancia de velocidad.

RECOMENDACIONES

- Estudiar otras formas de administrar el clúster disponibles como tecnologías para Apache Spark, para lograr un mejor equilibrio en la distribución de bloques y tareas entre los nodos trabajadores.
- Estudiar el uso de otros mecanismos para el procesamiento distribuido de datos distintos al uso de RDD, tales como: Dataframes o Dataset que muestran mejor rendimiento si se aplica cacheo en la memoria principal.
- Desarrollar soluciones paralelas para otras técnicas de estimación de parámetros basadas en estimadores bayesianos.
- Extender la propuesta de solución hacia modelos de ítems de crédito parcial y crédito parcial generalizado.

REFERENCIAS BIBLIOGRÁFICAS

Allamaraju, S. (2010). *RESTful Web Services Cookbook*: O'Reilly.

Andrea Gotzmann, P. D., & Louise M. Bahry, M. E. (2012). Software Review. jMetrik item analysis. *7*.

Antonov, A. (2015). *Spring Boot Cookbook* Packt Publishing.

Aroquipa, E. T. (2018). Sistemas distribuidos. *Portal UNAP*. Retrieved from http://www.unap.edu.pe/cidiomas/licing/pdf/sd.pdf

Aßmann, C., Gaasch, C., Pohl, S., & Carstensen, C. H. (2015). Bayesian estimation in IRT models with missing values in background variables. *57*(4).

Attiya, H., & Welch, J. (2004). *Distributed computing. Fundamentals, Simulations and Avanced Topics*: Willey Interscience.

Baker, F. B. (2001). The basics of Item Response Theory. University of Wisconsin.

Baker, R. S. J. D. (2010). *Data mining for education* (Vol. 7). Oxford, UK: Elsevier Ltd.

BAOSS, A. E. (Producer). (2015, 11 18). 10 Herramientas para manejar Big Data Analitics. Retrieved from https://www.baoss.es/10-herramientas-para-manejar-big-data-analytics/

Barcelona, U. d. (2017). Estimación de máxima verosimilitud. *Universidad de Barcelona*. Retrieved from http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo7/B0C7m1t10.html

Bock, R. D., & Aitkin, M. (1981). Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm. *46*(4).

Burke, B. (2014). *RESTful Java with JAX-RS 2.0*. Sebastopol: O'Relly Media Inc.

Camargo-Vega, J. J., Camargo-Ortega, J. F., & Joyanes-Aguilar, L. (2015). Conociendo Big Data. *24*(38).

Coursera. (2016). How the world learns. *Coursera*. Retrieved from http://coursera.tumblr.com/post/142363925112/introducing-our-new-infographic-how-the-world

Chanjin, Z., Xiangbin, M., Shaoyang, G., & Zhengguang, L. (2018). Expectation-Maximization-Maximization: A Feasible MLE Algorithm for the Three-Parameter Logistic Model Based on a Mixture Modeling Reformulation. *8*.

Chatzigeorgakidis, G., Karagiorgou, S., Athanasiou, S., & Skiadopoulos, S. (2017). FML-kNN: scalable machine learning on Big Data using k-nearest neighbor joins. *5*(1).

Chowdhury, M., Zaharia, M., & Stoica, I. (2014). Performance and scalability of broadcast in Spark. Retrieved from http://www. cs. berkeley. edu/~ agearh/cs267. sp10/files/mosharaf-spark-bc-report-spring10. pdf

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters: OSDI.

Dos Santos, R. P. (2015). Big Data: Philosophy, emergence, crowdledge, and science education. *8*(2).

Dropwizard, T. (2017). Getting Started. Dropwizard *Dropwizard* Retrieved from http://www.dropwizard.io/1.2.0/docs/getting-started.html

F. Fries, J., Witter, J., Rose, M., Cella, D., Khanna, D., & Morgan-DeWitt, E. (2014). Item response theory, computerized adaptive testing, and PROMIS: assessment of physical function. *41*(1).

Flink, A. (Producer). (2018a, 2/10/2018). Flink v1.4. Build Services. Retrieved from https://ci.apache.org/projects/flink/flink-docs-release-1.4

Flink, A. (Producer). (2018b, 1 2). Introduction to Apache Flink. Retrieved from https://flink.apache.org/introduction.html#features-why-flink

Frydenberg, M. (2015). Introducing Big Data Concepts in an Introductory Technology Course. (13).

Galindo, Á. J., & García, H. Á. (2010). Minería de Datos en la Educación. Madrid: Universidad Carlos III de Madrid.

Gil, A. A. (2015). Estimadores de máxima verosimilitud. *Universidad Carlos III en Madrid. Departamento de estadística*. Retrieved from http://halweb.uc3m.es/esp/Personal/personas/aarribas/esp/docs/estI_tema3.pdf

Goldin, I. (2015). Move your lamp post: Recent data reflects learner knowledge better than older data. *7*(2).

Güler, N., Uyanık, G. K., & Teker, G. T. (2014). Comparison of classical test theory and item response theory in terms of item parameters. *2*(1).

Guller, M. (2015). *Big Data Analytics with Spark*. New York.

Gupta, H. M. (2015). Integrated Services Digital Networks (ISDN): Structure and Services. *35*(1-2). doi: 10.1080/09747338.1994.11436449

Hambleton, R. K., & Swaminathan, H. (1985). *Item Response Theory: Principles and Applications*. New York: Springer Science.

Harwell, M. R., Baker, F. B., & Zwarts, M. (1988). Item parameter estimation via marginal maximum likelihood and an EM algorithm: A didactic. *13*(3).

Hsu, Y. (2000). On the Bock-Aitkin procedure—From an EM algorithm perspective. *65*(4).

Insua-Suárez, E., Fulgueira-Camilo, M., & Henry-Fuenteseca, V. (2015). Paralelización del Algoritmo Expectación-Maximización Utilizando OpenCL.

J. A. Larusson, B. W. (2014). *Learning Analytics. From research to practice*. United States.

Jach, A., & Alonso, A. (2017). Estimadores de máxima verosimilitud. *Universidad de Carlos III en Madrid. Departamento de estadística*. Retrieved from http://halweb.uc3m.es/esp/Personal/personas/amalonso/esp/e1tema4.pdf

Johnson, M. S. (2007). Marginal Maximum Likelihood Estimation of Item Response Models in R. *20*(10).

Kang, S. J., Lee, S. Y., & Lee, K. M. (2015). Performance Comparison of OpenMP, MPI, and MapReduce in Practical Problems.

L. Thorpe, G., & Favia, A. (2012). *Data Analysis Using Item Response Theory Methodology: An Introduction to Selected Programs and Applications.* The University of Maine: DigitalCommons.

Landset, S., Khoshgoftaar, T. M., Richter, A. N., & Hasanin, T. (2015). A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *2*(1).

Manyika, J., Chui, M., & Brown, B. (Producer). (2011, Mayo). Big data: The next frontier for innovation, competition, and productivity. *Digital Mckinsey*. Retrieved from http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation

Melià, D. J. L. (2017). Universidad de Valencia. *Métodos de Estimación de Parámetros en Teoría de la Respuesta al Ítem*. Retrieved from https://www.uv.es/~meliajl/Research/EstimationWeb/Fundamentos_archivos/frame.htm

Meyer, J. P. (2014). *Applied Measurement with jMetrik*. New York: Routledge.

Meyer, J. P. (2015). Feature Article. jMetrik: Open-source software for psychometric analysis

Meyer, J. P. (Producer). (2017, 12 24). Jmetrik. Psychomeasurement Systems. Software and Consulting Services. Retrieved from https://itemanalysis.com/jmetrik-download/

Natarajan, D. V. (2009). *Basic Principles of IRT And Application to Practical Testing & Assessment*: Asilomar: MeritTrac Services (P) Ltd.

Natesan, P., Nandakumar, R., Minka, T., & Rubright, J. D. (2016). Bayesian Prior Choice in IRT Estimation Using MCMC and Variational Bayes. *7*(1422).

Newman, S. (2015). *Building Microservices*. Sebastopol: O'Reilly Media.

Nielsen, F. (2016). *Introduction to HPC with MPI for Data Science*. Palaiseau, France: Undergraduate Topics in Computer Science. Springer.

ORACLE (Producer). (2017). Fork/Join. *The Java™ Tutorials*. Retrieved from https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html

Patsias, K., Rahimi, M., Sheng, Y., & Rahimi, S. (2012). Parallel Computing with a Bayesian Item Response Model. *2*(2).

Pedrosa, I., Juarros-Basterretxea, J., Robles-Fernández, A., Basteiro, J., & García-Cueto, E. (2015). Pruebas de bondad de ajuste en distribuciones simétricas, ¿qué estadístico utilizar? *, 14*(1).

Pentreath, N. (2015). *Machine Learning with Spark*. Birmingham - Mumbai: Packt Publishing Ltd.

Pérez Gil, J. A. (2011). Modelos de Medición: Desarrollos actuales, supuestos, ventajas e inconvenientes

Reyes-Ortiz, J. L., Oneto, L., & Anguita, D. (2015). Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. *53*.

Richard-Foy, J. (2014). *Play Framework Essentials* PACKT Books - Packt Publishing.

Richins, D., Ahmed, T., Clapp, R., & Reddi, V. J. (2018). Title Amdahl's Law in Big Data Analytics: Alive and Kicking in TPCx-BB (BigBench).

Rodriguez, A. (2015). RESTful Web services: The basics. *IBM*. Retrieved from https://www.ibm.com/developerworks/webservices/library/ws-restful/index.html?ca=dgr-jw22RESTfulBasics&S_Tact=105AGX59&S_CMP=GRsitejw22

Romero, C., & Ventura, S. (2012). Data mining in education. *3*(1).

Romero, C., Ventura, S., Pechenizkiy, M., Baker, & S.J.d., R. (2011). *Handbook of Educational Data Mining*. New York: Taylor and Francis Group, LLC.

Sande, B. v. d. (2013). Properties of the Bayesian Knowledge Tracing Model. *5*(2).

Schnabel, R. B., Koontz, J. E., & Weiss, B. E. (1982). A Modular System of Algorithms for Unconstrained Minimization. University of Colorado at Boulder: Computer Science Department.

Sehgal, G., & Garg, D. K. (2014). Improved Expectation Maximization Clustering Algorithm. *3*(5).

Sheng, Y., Welling, W. S., & Zhu, M. M. (2014). A GPU-Based Gibbs Sampler for a Unidimensional IRT Model.

Skala, K., Davidovi, D., Afgan, E., Sovic, I., & Sojat, Z. (2015). Scalable Distributed Computing Hierarchy: Cloud, Fog and Dew Computing. *2*(1).

Spark, A. (2017). MLlib is Apache Spark's scalable machine learning library. Retrieved from https://spark.apache.org/mllib/

SpringFox. (2017). Springfox Reference Documentation. Retrieved from https://springfox.github.io/springfox/docs/current/

Teng, F. (2014). A novel real-time scheduling algorithm. New York.

Trifu, M. R., & Ivan, M. L. (2014). Big Data: present and future. *V(1)*.

Turnquist, G. L. (2014). *Learning Spring Boot*. Birmingham, UK: Packt Publishing.

White, T. (2012). *Hadoop. The definitive guide, Third Edition*. Sebastopol: O'Relly.

Yadav, R. (2015). *Spark Cookbook*. Birmingham - Mumbai: Packt Publishing Ltd.

Yu, S., & Guo, S. (2015). *Big Data Concepts, Theories, and Applications*. New York Springer International Publishing Switzerland.

Yue, S., & Hambleton. R. K. (2017). Practical Consequences of Item Response Theory Model Misfit in the Context of Test Equating with Mixed-Format Test Data. 8(484).

Anexo 1. Resultados de las pruebas de rendimiento realizadas

Tabla 11. Resultados de las pruebas de rendimiento en el ambiente 2x2

Código	**2x2**				
Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Celeron CPU G1839 2.8GHz (2 núcleos)				2	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
85153	314279	98229	466328	151561	526264
80404	253091	107556	396759	150073	522450
71658	301030	98274	396300	155895	522240
71642	251638	98078	389192	144028	522025
70766	305586	98646	394637	139903	523170
82603	253557	97760	393344	149826	529061
71269	253536	97642	391309	153857	521727
71141	303926	108907	396720	170741	522783
72151	252082	101774	468014	162663	521718
71887	252089	98709	472830	147384	522303
68665	305271	103851	468062	155408	602914
78221	305262	98330	397125	161072	592852
70475	254209	98553	396044	143671	606901
73565	261377	98302	396187	159942	522270
69855	309392	99504	474378	160683	521977
70183	256528	98125	391221	149567	523518
70161	253229	98553	395532	149551	522965
69787	262047	99455	396842	146709	520535
69422	255320	97757	466661	148746	600709
68121	255496	98481	447973	151569	597636

Tabla 12. Resultados de las pruebas de rendimiento en el ambiente 1x4

Código	**1x4**
Datos del entorno de pruebas	
Datos de los nodos trabajadores	**Cantidad**
Intel Core i3-4160 3.6GHz (4 núcleos)	1

Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
68719	227502	86468	356268	127087	464448
61853	226696	86624	355666	124746	461865
60811	226828	88284	355888	124939	456446
60565	226789	86556	355116	126064	456268
60725	226545	86646	356165	121711	461766
60638	227023	86846	355223	117848	454752
60871	227417	88109	355166	123500	454059
60437	227243	86477	355820	129491	453346
60226	227321	86815	355337	130583	452753
60598	227192	87027	356949	120812	451959
60530	227519	86976	355299	123554	452685
60781	227154	87018	355985	125447	453045
62377	227638	88331	355704	129991	452851
60526	228206	86499	356152	121762	453182
60665	227458	87024	356795	121941	452650
60769	227098	87120	355495	125878	453217
60809	228314	88081	356348	130558	452484
60856	227415	86922	355946	124533	452781
61356	227560	86635	356428	122872	452457
62206	227454	87012	356148	119399	453873

Tabla 13. Resultados de las pruebas de rendimiento en el ambiente 1x2+1x4

Código	**1x2+1x4**				
Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Core i3-4160 3.6GHz (4 núcleos)				1	
Intel Celeron CPU G1839 2.8GHz (2 núcleos)				1	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
59632	191391	64472	363537	104135	369592
51893	187718	70500	300109	116662	357443
53187	189158	73127	331347	121710	361235
51280	175557	76146	329655	136491	359264

50462	174368	78466	359393	123795	356185
51564	178114	85431	340924	134846	360536
49894	187355	78524	339474	125243	357040
51508	175764	80821	347021	133278	371089
48199	178321	80268	341033	127864	369895
47754	185784	79497	337862	124196	362714
49451	172801	80093	348474	131762	369240
53456	173168	86664	349933	124889	370376
54507	171641	79768	378949	122265	359639
56641	172852	78946	352104	129087	360930
53628	175337	77770	359654	137464	360421
56062	180567	90831	359201	140626	370595
58144	175464	82203	414130	132243	362708
55402	175574	79135	355464	132221	365267
58020	174700	82235	346735	130952	361255
59752	176744	80987	354248	132566	360032

Tabla 14. Resultados de las pruebas de rendimiento en el ambiente 2x2+1x4

Código	**2x2+1x4**				
Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Celeron CPU G1839 2.8GHz (2 núcleos)				2	
Intel Core i3-4160 3.6GHz (4 núcleos)				1	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
60203	212495	68962	274267	103787	352775
67762	181120	72000	287412	109376	368728
57911	198993	72887	287185	99429	362081
50235	186807	72405	263827	100941	359350
55854	175154	71895	276252	96912	366315
50271	205713	68545	264301	100620	397764
50772	203032	70221	265368	109786	369516
49577	187852	69863	277540	102406	361430
50576	161366	71904	280634	108219	388396
55358	158773	77205	267376	101529	377883

50630	165111	72840	271521	111296	353006
54394	158851	69628	291047	105136	362122
50976	164509	67804	284697	96001	357350
57626	169238	71760	268171	97955	359630
50388	197973	69310	263794	101162	379008
48750	172354	70198	263272	107946	356294
50320	169272	69536	277391	106502	364974
61628	162956	69178	304652	103253	383825
56406	172422	74636	267457	104428	354918
50332	170756	76600	263764	105531	359849

Tabla 15. Resultados de las pruebas de rendimiento en el ambiente 2x4

Código	**2x4**				
Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Core i3-4160 3.6GHz (4 núcleos)				2	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
50050	158450	63965	217466	84609	283847
41926	147719	58310	249414	89735	323013
40426	160085	57560	252618	95432	341287
44893	162422	52406	267863	90320	363054
43452	184746	54836	259703	96377	369185
42388	172463	57491	261618	96483	364476
41865	163726	53997	244160	87966	323212
44705	168016	58020	265683	90636	336064
40062	167715	60351	254822	87852	350204
45094	168067	59564	251591	102900	326773
40475	164996	59908	263663	105539	352921
46181	170120	63838	253433	92603	326907
40739	167685	62503	254097	100092	355380
39949	168146	64054	250576	119995	367960
41486	166431	60689	262912	90968	313186
40459	167651	64430	270269	90079	378408
42258	186143	62682	259418	91559	330045

41948	162024	60913	260973	105529	363621
44584	165819	62237	249772	101670	321668
44132	168713	62310	266804	89868	386814

Tabla 16. Resultados de las pruebas de rendimiento en el ambiente 2x4+1x2

Código	**2x4+1x2**				
Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Core i3-4160 3.6GHz (4 núcleos)				2	
Intel Celeron CPU G1839 2.8GHz (2 núcleos)				1	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
44105	185687	80615	216240	73851	263418
44604	138120	58688	255983	82529	294220
43084	139155	51698	353090	88061	542416
48095	142688	62318	217871	85938	423134
41914	148786	60168	215040	83124	342699
51737	155877	57952	247935	96266	334277
41286	143431	51504	212696	85855	289664
43158	150141	56439	215827	76644	310791
42025	162125	54326	243091	80704	291427
43532	158750	58069	210220	88205	296446
43660	164092	60366	253186	86607	283302
43679	143690	54614	244857	87478	330578
45218	136209	54990	253949	90243	359097
45054	142766	60140	239214	88962	348864
45449	151414	54239	217459	83819	315449
42603	149876	63695	246036	103069	357885
41387	163527	61432	226538	85077	305777
45015	159342	58494	234125	82654	290071
44596	137885	63632	206801	92132	305276
44015	139091	58574	257811	89105	309224

Tabla 17. Resultados de las pruebas de rendimiento en el ambiente 3x4

Código	**3x4**

Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Core i3-4160 3.6GHz (4 núcleos)				3	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
53684	127618	62469	217955	70910	299107
46130	124975	56880	186154	68280	278636
41060	125016	48635	179780	72936	274502
40981	126175	51965	171371	80658	228948
42793	130985	53177	181311	82398	294070
39906	132249	48747	204100	70326	226898
40063	132813	51371	217110	79407	230905
40242	131873	46883	184822	82094	250808
39838	143014	58240	210746	71727	224135
43078	136616	47731	212311	82142	271164
40148	137467	51413	165902	68121	301411
39290	128101	51893	169026	72811	300488
39468	132242	50369	167662	74200	286341
41045	144813	49894	215684	82482	283893
41980	134391	52213	210428	79467	255897
39725	135442	47005	209484	75541	235828
40546	147137	48312	210640	83134	296128
41461	137870	54653	206459	75312	235187
39342	129682	47128	215509	74639	290993
39737	137782	49251	215411	87400	293242

Tabla 18. Resultados de las pruebas de rendimiento en el ambiente 2x4+2x2

Código	**2x4+2x2**				
Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Core i3-4160 3.6GHz (4 núcleos)				2	
Intel Celeron CPU G1839 2.8GHz (2 núcleos)				2	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
61288	142450	61270	248624	78452	323508

48092	139737	62069	261825	85809	352467
49117	133689	61505	254234	89786	425962
46307	131121	63137	250188	85658	265517
58199	154650	59874	253121	82673	291337
45884	139261	56984	261820	112494	307446
44907	137119	57626	215133	92367	301479
47361	147820	58270	251243	114069	289013
48348	142894	61624	220146	88245	292072
44584	135412	61484	214365	74408	280352
45887	141913	60579	198288	85402	333540
45018	141433	63205	199896	89394	346943
54130	158337	53092	230725	87949	302832
44082	148407	52639	264694	93313	329653
53019	146402	71504	227743	79146	319689
45542	155740	61955	237996	80818	319517
46922	136721	56260	270369	118174	403868
46604	143943	59727	246378	93824	305544
43436	143266	62564	206966	84627	301198
50814	163174	54671	276196	80717	331394

Tabla 19. Resultados de las pruebas de rendimiento en el ambiente 3x4+1x2

Código	**3x4+1x2**				
Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Core i3-4160 3.6GHz (4 núcleos)				3	
Intel Celeron CPU G1839 2.8GHz (2 núcleos)				1	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
55626	133620	58073	201505	81248	301153
46296	133012	58179	187483	82354	269334
44725	136150	49886	176616	80030	272926
48158	133741	58633	165214	69323	330844
45379	131895	57608	215821	83929	258799
43190	134226	57237	206331	83141	261451
46543	130403	57868	206648	80122	326512

49009	132072	54737	186701	74920	325751
43953	140692	54372	198539	94482	282722
44581	138020	59652	208184	72751	263774
44396	135032	53602	190022	81548	263744
46876	134976	57191	195692	82095	223255
41482	132322	59301	167242	91761	268709
45366	147576	57896	197752	80942	314850
47341	137218	59738	191934	78138	257356
43996	134348	51119	191257	73738	259165
44237	136970	62534	176351	67961	233556
44888	145214	63699	178183	78347	259463
44837	141723	65098	199864	81695	261526
45056	136670	61006	188943	73629	266891

Tabla 20. Resultados de las pruebas de rendimiento en el ambiente 3x4+2x2

Código	**3x4+2x2**				
Datos del entorno de pruebas					
Datos de los nodos trabajadores				**Cantidad**	
Intel Core i3-4160 3.6GHz (4 núcleos)				3	
Intel Celeron CPU G1839 2.8GHz (2 núcleos)				2	
Tamaño de la entrada					
2000x50	**2000x200**	**5000x50**	**5000x200**	**10000x50**	**10000x200**
58526	136250	51767	205766	83582	291209
49425	136304	57178	186318	84487	256295
56789	143215	58466	164654	72379	268629
48787	139707	52084	165334	108513	320109
47902	146743	52010	181122	85173	248198
51383	144736	51522	195706	73962	279110
48966	150056	57579	195672	76325	265309
47115	140910	51195	199748	76891	248285
47792	149991	71357	234680	72079	254470
46224	141396	51229	200007	79395	269826
49793	140233	49899	194333	75315	264921
48857	137063	53009	197199	78516	272652
45770	137433	52847	164655	82313	268814

47499	133496	58239	189588	81590	268324
48130	139676	57224	192104	81461	225000
49164	138624	57851	185483	80051	251374
47610	140701	58793	187005	76279	222418
51129	135203	59855	163006	78318	261238
46855	137123	58097	213534	76118	253528
49609	135944	53210	182705	98354	314272