UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

SOLUCIÓN DISTRIBUIDA PARA DISMINUIR EL TIEMPO DE EJECUCIÓN EN LA DETECCIÓN DE PLAGIO EN JUECES EN LÍNEA DE PROGRAMACIÓN

Tesis presentada en opción al título de Máster en Informática Aplicada

Autor: Ing. Leandro González Vallejo

Tutor: DrC. Rafael Arturo Trujillo Rasúa

Co-tutor: MsC. Tomás Orlando Junco Vázquez

La Habana, diciembre de 2014

"Año 56 de la Revolución"

Agradecimientos

Agradezco al Movimiento de Programación Competitiva Tomás López Jiménez donde surgió este tema de investigación, en especial a Dovier como el máximo responsable de los recientes logros de Cuba en las competencias de la ACM-ICPC.

A toda una lista de colegas, profesores y amigos que formaron parte directa del desarrollo de esta investigación aportándome conocimientos, ideas, críticas y revisiones. La lista es grande pero espero que ellos se vean identificados.

Agradezco a mis tutores que fueron mi guía en años de investigación.

En especial agradezco a mi señora, a Yisel, por varias páginas llenas de razones.

Dedicatoria

A mi familia, por siempre confiar en mí.

A mis tutores, los que están en la portada y los que no, por su indispensable apoyo.

A mi señora, porque siempre estuvo a mi lado, dándome ánimo y ayuda.

Declaración de Autoría

Yo, Leandro González Vallejo, con carné de identidad 87062625863, declaro que soy el autor principal del resultado que expongo en la presente memoria titulada: “Solución distribuida para disminuir el tiempo de ejecución en la detección de plagio en jueces en línea de programación”, para optar por el título de Máster en Informática Aplicada y 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.

Este trabajo fue desarrollado en el período 2012-2014 en estrecha colaboración con Yisel Quiñones Guerrero, Rafael Arturo Trujillo Rasúa, Tomás Orlando Junco Vázquez y Enrique José Altuna Castillo; quienes me reconocen la autoría principal del resultado expuesto en esta memoria.

Declaro que todo lo anteriormente expuesto se ajusta a la verdad, y asumo la responsabilidad moral y jurídica que se derive de este juramento profesional.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Autor: Ing. Leandro González Vallejo

Resumen

En los jueces en línea de programación se han detectado casos de plagio y esta es una actitud vergonzosa que debe evitarse y combatirse. Las soluciones identificadas para la detección de plagio no tienen en cuenta algunas informaciones presentes en los jueces en línea que pueden aprovecharse para emitir un dictamen más exacto. La única herramienta que se logró identificar para la detección de plagio en jueces en línea, tiene altos tiempos de ejecución y no cumple las exigencias de estas aplicaciones, que tienen que procesar una considerable cantidad de soluciones enviadas por los programadores. Para disminuir el tiempo de ejecución, se desarrolló una solución basada en la distribución del procesamiento y el empleo de un sistema de caché distribuido. Para la detección de plagio se propuso la mezcla ponderada de múltiples criterios, entre los que son imprescindibles la estructura del código fuente y la clasificación del problema. La solución se aplicó en el Juez en Línea Caribeño utilizando el *middleware* de mensajería RabbitMQ para la distribución del procesamiento, el sistema de caché distribuida Redis, el algoritmo para la detección de plagio *Greedy String Tiling* y un método para la desestimación del código fuente correspondiente a la implementación de algoritmos clásicos dada la clasificación del problema. Empleando un clúster *Beowulf* con 16 computadoras, se logró disminuir los tiempos de ejecución en un 91.35% respecto a la ejecución de los mismos algoritmos pero sin distribuir el procesamiento.

Palabras clave: caché distribuida, computación distribuida, detección de plagio, juez en línea.

Índice

Introducción

La Programación, en el contexto informático, es el proceso de diseñar, codificar, depurar y mantener un conjunto de instrucciones destinadas a resolver un problema determinado mediante el uso de la computación (1). Unido a otras disciplinas como la Ingeniería de Software, ha permitido el desarrollo de programas que en menos de un siglo pasaron de simples procesamientos estadísticos a la gestión de grandes cantidades de información y complejos cálculos. Bjarne Stroustrup, creador del exitoso lenguaje de programación[1] C++, afirmó: "La única forma de aprender a programar es programando" (2), por lo que en todo el mundo especialistas de la Informática dedican una considerable cantidad de tiempo a practicar e intentan aumentar sus conocimientos de determinado lenguaje de programación y sus habilidades para escribir algoritmos[2].

Un juez en línea de programación es una aplicación web con un conjunto permanente de problemas que deben ser resueltos mediante la Programación. Su principal función es evaluar automáticamente los intentos de solución de los programadores. Existen jueces en línea de varios tipos, en el contexto de este trabajo cuando se use el término juez en línea, se estará haciendo referencia a un juez en línea de programación. Estas aplicaciones ganan popularidad como plataforma para el entrenamiento desde el año 1997 con la publicación del juez en línea de la Universidad de Valladolid (UVA) en España, creado por el profesor de matemáticas Miguel Ángel Revilla (3). El UVA junto al Sphere (SPOJ) de la Universidad de Gdansk en Polonia y al PKU (POJ) de la Universidad de Peking en China, son tres de los jueces en línea más representativos en el mundo.

En la Universidad de las Ciencias Informáticas (UCI) se desarrolla una línea de investigación relacionada con las herramientas de apoyo al proceso de enseñanza aprendizaje, particularmente en las asignaturas de programación con los jueces en línea (4). Estas aplicaciones contribuyen al aumento significativo que ha experimentado el movimiento de programación competitiva desde el año 2009. Se han celebrado diversas competencias, programas de entrenamiento, se percibe un creciente interés de los estudiantes por el tema

[1] Un **lenguaje de programación** es un idioma artificial diseñado para la comunicación hombre-máquina.
[2] Un **algoritmo** es una serie finita de pasos para resolver un problema (55).

y se insertó la UCI en el movimiento Internacional de Competencias de Programación Inter-Colegios de la Asociación para Máquinas Computadoras (ACM-ICPC).

Durante el año 2006, surge en la entonces Facultad 8 de la UCI[3] el primer juez en línea de esta institución, desarrollado por la iniciativa *Xtreme Programming*[4] (Xtreme) que llegó a contar con 2699 usuarios registrados que enviaron 79220 intentos de solución a 662 ejercicios y 69 competencias (5). En el año 2007 se publicó el juez en línea de la Cátedra de Programación Avanzada (CPAV) con estadísticas relevantes de 6527 usuarios, 163119 envíos, 711 problemas y 157 competencias realizadas (6).

En este contexto surge el Juez en Línea Caribeño (COJ por sus siglas en inglés de *Caribbean Online Judge*) convirtiéndose en la plataforma principal de entrenamiento para los programadores. Los resultados de este proyecto trascienden las fronteras cubanas, incluso de la región caribeña. Desde su publicación el 5 de junio del 2010, se han registrado 17004 usuarios de 850 instituciones pertenecientes a 156 países, se han enviado 694283 intentos de solución y fueron celebradas 380 competencias (7).

Pero los administradores del COJ han detectado un número considerable de soluciones similares, incluso algunas idénticas. El plagio es una violación grave que puede ser reflejada en diferentes esferas. La Real Academia Española al definir la palabra plagio, nos remite directamente a la acción y efecto de plagiar, entendiendo esta última como “copiar en lo sustancial obras ajenas, dándolas como propias” (8). Un motivo para cometer plagio es la intención de hacer más rápido y con menos esfuerzo determinada tarea, sin valorar que moralmente no es correcto ni honesto apropiarse de ideas ajenas. En este sentido la ley contempla algunos tipos de plagio como delito y es sancionado de acuerdo a su gravedad.

En el juez en línea de la Cátedra de Programación Avanzada de la UCI se desarrolló un sistema basado en la comparación de texto plano, que aunque no tenía en cuenta la semántica ni otros datos estructurales de las soluciones, sirvió para alertar sobre la gravedad del asunto. El 9 de septiembre del 2009 la primera ejecución de dicho sistema detectó 948 posibles casos de plagio correspondientes a 374 usuarios, en una

[3] La entonces **Facultad 8** es actualmente la Facultad 4, en el año 2010 la UCI cambia por cuestiones organizativas la constitución de sus facultades.
[4] La Iniciativa ***Xtreme Programming*** o simplemente *Xtreme* surge en la Facultad 8 de la UCI con el objetivo de promover la programación competitiva.

segunda ejecución el 31 de mayo del 2010 fueron 271 los casos detectados y 211 los usuarios involucrados (6).

En el sitio oficial del concurso español de programación “ProgramaMe” se especifica entre las reglas de participación: “Los participantes garantizarán ser los autores de todo el código que envíen al Juez durante el concurso. La detección de plagios supondrá la descalificación automática del equipo o equipos implicados” (9).

El plagio en jueces en línea está motivado, entre otras razones, por el entorno competitivo en que los programadores desean mejores lugares en el ranking y la baja capacidad de detección que permite a los usuarios cometer estos hechos impunemente. Solo se ha logrado identificar cuatro jueces en línea que se esfuercen de forma planificada en detectar plagio, ninguno de ellos cuenta con una solución automatizada y eficaz que le permita realizar esta tarea.

Atendiendo a esta problemática, en el año 2012 se desarrolló un módulo para la detección de plagio en el Juez en Línea Caribeño. Este módulo estaba basado en una mezcla entre comparación del código fuente mediante el algoritmo *Greedy String Tiling,* el análisis de las características del problema y del perfil del usuario. Los resultados obtenidos con esta investigación fueron favorables y fue la primera solución para la detección de plagio desarrollada específicamente para un juez en línea de programación. No obstante, los tiempos de ejecución eran altos e impedían que su uso se generalizara más allá de la confirmación de una sospecha de plagio entre algunos cientos de soluciones; por ejemplo para detectar plagio en una competencia (10).

Por otra parte, el modelo arquitectónico en forma de módulo afectaba directamente los recursos computacionales del COJ y perjudicaba la ejecución de otros procesos en el Juez, algunos incluso más importantes como la evaluación de las soluciones. Además, el estar atado a la arquitectura y tecnologías del Juez en Línea impedía que su uso se extendiera a otras aplicaciones en las que también está presente esta problemática.

Es necesario mejorar el proceso de detección de plagio en los jueces en línea, pero no se ha logrado identificar alguna herramienta para la detección que tenga tiempos de ejecución favorables y se adecue a las características de un juez en línea. Las herramientas existentes utilizan algoritmos con una alta complejidad computacional y centran su funcionamiento en la comparación del resultado generado a partir

del análisis léxico[5] del código fuente[6]. Además, obvian características importantes como la clasificación del problema. Es diferente el análisis que debe hacerse cuando la solución requiere del uso de un algoritmo clásico que implica un bloque de código que es lógico que sea similar.

Una forma de disminuir el tiempo de ejecución de una aplicación es compartiendo el procesamiento entre múltiples ordenadores. Una solución distribuida es una "solución informática con componentes hardware y software, que están en ordenadores conectados en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor" (11). El COJ no cuenta con un mecanismo que permita distribuir el procesamiento.

A partir del análisis anterior se concluye que:

- Se ha detectado plagio en varios jueces en línea, incluyendo el Juez en Línea Caribeño.
- Cometer plagio es una actitud inmoral y deshonesta que no debe quedar impune.
- Solo se ha logrado identificar una herramienta adecuada para la detección de plagio en jueces en línea pero sus tiempos de ejecución son altos.

Atendiendo a la anterior problemática, se identifica el siguiente **problema científico:** ¿Cómo disminuir el tiempo de ejecución al detectar plagio en jueces en línea de programación?

Se plantea como **objeto de estudio** de la investigación el proceso de detección de plagio en código fuente.

El **objetivo general** de la investigación es desarrollar una solución distribuida para disminuir el tiempo de ejecución al detectar plagio en jueces en línea de programación.

Para darle cumplimiento al objetivo general se plantean los siguientes **objetivos específicos:**

1- Elaborar el marco teórico sobre el plagio como problema, sus antecedentes, herramientas para la detección, metodologías y tendencias en su desarrollo.
2- Caracterizar los jueces en línea de programación en relación a sus insuficiencias, limitaciones y desventajas que dan lugar a la necesidad de dar solución a la problemática planteada.

[5] El **análisis léxico** es la primera fase de un compilador en que se convierte el código fuente en una secuencia de componentes léxicos.
[6] El **código fuente** es el conjunto de líneas de texto con instrucciones escritas en un lenguaje de programación.

3- Desarrollar una solución distribuida que permita disminuir el tiempo de ejecución al detectar plagio en jueces en línea de programación.
4- Validar la solución propuesta a partir de su aplicación en el Juez en Línea Caribeño y la realización de experimentos que midan la disminución en los tiempos de ejecución.

Se plantea como **hipótesis científica** que una solución distribuida disminuirá el tiempo de ejecución al detectar plagio en jueces en línea de programación.

Tabla 1: Variables

	Nombre	Definición	Dimensión	Indicador
Independientes	Solución distribuida	Solución informática con componentes hardware y software, que están en ordenadores conectados en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor (11).	Cantidad de computadoras	$unidades$
Dependientes	Tiempo de ejecución al detectar plagio en jueces en línea de programación	Tiempo que demora en un juez en línea el proceso de encontrar una solución de referencia ***R*** y una solución sospechosa ***S*** tales que la programación de **S** se realizó parcial o totalmente basada en ***R.***	Tiempo empleado en realizar una detección de plagio	$\frac{Detecciones}{Segundo}$

El **resultado esperado** es una solución distribuida para disminuir el tiempo de ejecución al detectar plagio en jueces en línea de programación.

Novedad

La investigación propone el uso de tecnologías utilizadas en la computación de alto rendimiento que permiten el procesamiento distribuido y no se han empleado con anterioridad en la detección de plagio. Además, de acuerdo a la investigación realizada, no existen detectores de plagio desarrollados

específicamente para aprovechar las características de los jueces en línea de programación y que alcancen bajos tiempos de ejecución.

Aporte

La investigación tiene un marcado aporte social, al facilitar la detección de plagio que constituye un problema ético que afecta particularmente el entorno académico. La presencia de este problema en los jueces en línea de programación, afecta la preparación que deben adquirir los estudiantes que practican en estos sistemas los conocimientos adquiridos en carreras afines con la computación.

Para la realización de esta investigación se utilizaron los siguientes **métodos científicos**:

Métodos teóricos:

- Analítico-sintético: para el análisis de la teoría, documentación y demás antecedentes relacionados con el proceso de detección de plagio en código fuente, que permitió concluir las características principales que debe tener la solución propuesta, así como las herramientas y tecnologías óptimas para su desarrollo.
- Histórico-lógico: para estudiar la evolución histórica de los sistemas para la detección de plagio y caracterizar el avance en las tecnologías utilizadas para su desarrollo, particularizando en el área correspondiente a la detección de plagio en código fuente.
- Modelación: para representar los procesos, datos, características inherentes al campo de acción y el diseño de la solución.

Métodos empíricos

- Observación: para entender mejor el fenómeno, interactuando con él de forma sistémica, tanto interna como externa. La observación se realizó durante los períodos en que el autor fue concursante y parte del grupo analizado en la presente investigación y luego bajo un nueva perspectiva como entrenador.
- Encuesta: para determinar el criterio del personal implicado sobre la necesidad de detectar el plagio, su importancia, nivel de dificultad y las acciones post-detección que se deben llevar a cabo con los usuarios que cometan plagio. Además de proveer una vía por la cual obtener nuevas ideas sobre cómo abordar el problema.

- Experimentación: para la demostración de la hipótesis a partir de pruebas que confirmen que se disminuye el tiempo de ejecución en la detección de plagio en jueces en línea, a partir de la solución desarrollada.

Técnicas cualitativas:

- Técnica de Iadov: para conocer el nivel de satisfacción de los usuarios y administradores del Juez en Línea Caribeño con la solución implementada.

El documento está organizado en tres capítulos que detallan la investigación previa, selección de algoritmos para la detección, esquema de solución, implementación, aplicación de las tecnologías seleccionadas y la validación de la solución propuesta verificando la disminución de los tiempos de ejecución; quedando la **estructura capitular** de la siguiente manera:

Fundamentación teórica**:** se documentan los principales conceptos relacionados con el plagio y las herramientas detectoras. Se analizan las aplicaciones más comunes de este tipo de sistemas y los algoritmos que las respaldan. Se explica el objetivo y funcionamiento de los jueces en línea, ejemplificando con el Juez en Línea Caribeño. Se identifican las herramientas y tecnologías necesarias para el desarrollo exitoso de la investigación.

Propuesta de solución**:** se desarrolla una solución para disminuir el tiempo de ejecución a partir de la distribución de las detecciones y la aplicación de un sistema de caché distribuida para acceder a los códigos fuentes analizados. Se definen los algoritmos a utilizar en la detección de plagio en jueces en línea.

Implementación y validación**:** se valida la propuesta a partir de su implementación en el Juez en Línea Caribeño. Se caracteriza la capacidad de procesamiento mediante experimentos. Se aplica la técnica de Iadov para determinar el nivel de satisfacción de los encuestados con la solución desarrollada. Se comprueba el cumplimiento de los requisitos con pruebas funcionales a partir del método de caja negra y la técnica partición equivalente.

Fundamentación teórica

En el presente capítulo se realiza un estudio del estado del arte de la detección de plagio en código fuente. Los objetivos principales son caracterizar la problemática, analizar el componente ético que convierte a la práctica del plagio en una actitud negativa, identificar los principales algoritmos y herramientas utilizadas para la detección, caracterizar a los jueces en línea e investigar en ellos la presencia y detección de plagio a partir de herramientas existentes. Por último, se identifican las tecnologías necesarias para disminuir el tiempo de ejecución en la detección de plagio en jueces en línea.

1.1 Sobre el plagio

El plagio es una violación grave que puede ser reflejada en diferentes esferas de la sociedad. La Real Academia Española al definir la palabra plagio, nos remite directamente a la acción y efecto de plagiar, entendiendo esta última como “copiar en lo sustancial obras ajenas dándolas como propias” (8).

El *Oxford English Dictionary* define plagio como “La acción o la práctica de plagiar; la apropiación ilícita o el robo, y publicación como propios, de las ideas, o de la expresión de las ideas (literario, artístico, musical, mecánico, etc.) de otros” (12).

El delito del plagio viola los derechos de autor y paternidad que rodean la creación de una obra. El derecho de autor es la protección que le otorga el Estado al creador de las obras literarias o artísticas desde el momento de su creación y por un tiempo determinado. Por otro lado, se define como propiedad intelectual una disciplina normativa que protege las creaciones intelectuales provenientes de un esfuerzo, trabajo o destreza humana, dignos de reconocimiento jurídico. La propiedad intelectual comprende: el derecho de autor, los derechos conexos[7] y la propiedad industrial (13).

Actualmente el auge de Internet ha hecho que sea más fácil plagiar el trabajo de otros (14). En una encuesta realizada en el *Pew Research Center* en Estados Unidos el 89% de los encuestados indicó que “las

[7] **Derechos conexos** es un término relacionado con la ley de derechos de autor y *copyright*, para referirse a derechos similares, los derechos conexos son parte del marco jurídico del *copyright*.

computadoras e Internet han jugado un papel muy importante" (15), en el incremento sustancial que ha tenido el plagio en las Universidades. Es común que muchos sitios copien y peguen información sin la autorización correspondiente, engañando a sus lectores al presentar los datos como propios. De igual manera, las personas copian y pegan de sitios e investigaciones científicas sin referenciar la fuente, a estas acciones se les conoce como plagio de texto (16); en la Tabla 2 se exponen algunas estadísticas al respecto. Un caso particular lo constituye el plagio de algoritmos y códigos fuentes. Sobre todo en el ámbito académico el plagio es un comportamiento inaceptable y actualmente existen varias herramientas informáticas que permiten detectarlo.

Tabla 2: Porcentaje de estudiantes que admiten el haber realizado plagio en un trabajo escrito (17)

Acción	Pregrado	Postgrado
Parafrasear o copiar de una fuente escrita algunas oraciones sin referenciarlo	38%	25%
Parafrasear o copiar de Internet algunas oraciones sin referenciarlo	36%	24%
Fabricar o falsificar una bibliografía	14%	7%
Entregar un trabajo copiado de otra persona	8%	4%
Copiar casi palabra por palabra de una fuente escrita sin citarlo	7%	4%
Obtener un artículo de un sitio de Internet de venta de artículos	3%	2%

1.2 Generalidades de los sistemas para la detección de plagio

Los sistemas para la detección de plagio son herramientas con tecnologías informáticas que realizan la búsqueda de similitudes entre muestras con el fin de encontrar fragmentos plagiados parcial o totalmente. Se basan en diversas técnicas de recuperación y extracción de información así como de reconocimiento de formas y teorías de la información (14). Estos sistemas comienzan a surgir a mediados de los años 70 como posible solución ante la inaceptable tendencia presente en investigaciones científicas, obras, algoritmos y códigos de programación (18).

1.2.1 El problema de la detección de plagio

El problema de la detección automática de plagio puede verse como un problema de búsqueda y/o clasificación. Los elementos de referencia ***D*** y el elemento sospechoso ***S*** son suficientes para describir el planteamiento de la detección de plagio:

Sean ***S*** un elemento sospechoso y ***D*** un conjunto de elementos de referencia, el objetivo de la detección automática de plagio es encontrar aquel elemento $d \in D$ que haya sido utilizado como fuente para obtener el elemento ***S***, el cual presumiblemente es un caso de plagio. Dicha búsqueda puede llevarse a un nivel más específico: Sea $S_i \in S$ un fragmento plagiado, el objetivo es encontrar aquel fragmento $d_j \in d$ tal que d_j es la fuente del fragmento plagiado S_i (14).

1.3 Aplicaciones de los sistemas para la detección de plagio

Los sistemas para la detección automática de plagio tienen aplicación en diferentes esferas de la vida profesional, a continuación se hará referencia a su utilización en el entorno académico, empresarial y el derecho de autoría.

1.3.1 Los sistemas para la detección de plagio en el entorno académico

El tema del plagio académico se refleja en los trabajos encargados a los alumnos, es actualmente un problema importante y siempre lo ha sido. En 1995 Alschuler y Blimling hablaban de epidemia refiriéndose al aumento en esta tendencia (19); pero quizás es más común ahora, debido a la colosal cantidad de información disponible en Internet y en general a las facilidades que proporcionan las nuevas tecnologías de la información y las comunicaciones (TIC). En España el 61.1% del alumnado universitario afirma haber incorporado en sus trabajos fragmentos de páginas web, como si fueran propios, al menos una vez; estudios confirman la práctica generalizada de copiar y pegar sin citar las fuentes (19). El plagio de ensayos, seminarios, artículos y prácticas de programación es un problema serio en la enseñanza actual. Los profesores no se dan cuenta en algunas ocasiones que el trabajo de sus estudiantes no es del todo honesto, debido a que la detección manual es complicada y en ocasiones los pedagogos no cuentan con el tiempo suficiente para detectar la situación.

Si cierto es que la tecnología ha ayudado de una forma u otra a facilitar el plagio, no se puede descartar que a la vez ha ayudado a prevenirlo y a detectarlo porque existen herramientas informáticas desarrolladas con este fin.

Como es tarea de todos eliminar estas incidencias, varias universidades del mundo han creado sistemas de este tipo para combatir el plagio en sus centros de estudios y brindar sus servicios a otras esferas y entidades, entre estos sistemas se encuentran:

- MOSS, en la Universidad de Stanford, EUA.
- JPlag, en la Universidad de Karlsruhe, Alemania.
- YAP3, en la Universidad de Sydney, Australia.

En las universidades británicas y norteamericanas, por ejemplo, existe un protocolo por el que debe pasar cualquier trabajo de sus estudiantes. Entre los programas más utilizados con este objetivo están: Turnitin, Copycatch y Eve2 (20).

1.3.2 Los sistemas para la detección de plagio en el entorno empresarial

Con el arribo de la Web 2.0, las empresas han entrado a una nueva era de la informatización, ahora no están ajenas a las tendencias actuales y a los beneficios de Internet. Las que se han apoyado en el *marketing* a través de los medios digitales, han obtenido resultados satisfactorios en el mundo competitivo y empresarial. El fácil acceso a la información que brinda la Red ha favorecido un aumento, en cierta medida, de los casos de plagio entre empresas. Esto conlleva a que otras empresas les copien productos, servicios e ideas, por lo que deben protegerse ante aquellas plagiadoras que se valen del esfuerzo y resultados de otros para atribuirse méritos ajenos (21).

Existen también herramientas para la detección automática de plagio disponibles para que las empresas se protejan. WCopyFind es un software desarrollado por Bloomfield de la Universidad de Virginia, Estados Unidos, en el año 2004 y especializado en la detección de plagio en el entorno empresarial (21).

1.3.3 Los sistemas para la detección de plagio en la protección al derecho de autoría

El plagio constituye el más grave atentado al derecho de autor, pues en esencia significa desconocer la paternidad del autor, y por consiguiente, la relación que le une con la obra sustrayéndole todo conocimiento e ignorándole toda aportación creativa (22).

Actualmente las investigaciones científicas, artículos, libros, literatura, páginas webs que se encuentran en la red, se ven amenazadas por personas que no respetan los derechos de autor y se atribuyen estas creaciones como propias. En ocasiones consientes del delito, pero en otras, es por desconocimiento de la cultura de referenciar y citar a los legítimos autores. Existen, hoy día, formas más avanzadas de proteger los derechos de autoría haciendo uso de la tecnología. Los sistemas de detección automática de plagio se crean para que cumplan la función de identificar las obras que han sido plagiadas en su totalidad o fragmentos de las mismas.

A continuación se mencionan algunos programas que se encargan de detectar plagios en libros, literatura de todo tipo, imágenes y audio:

- Para detectar el plagio en texto el sistema Viper basa su funcionamiento en buscar entre los documentos locales y en Internet. Existen otros con este mismo objetivo como PlagiarismChecker, Plagium, Chimpsky o CopyTracker.
- Para identificar plagio en las imágenes GazoPa y TinEye, aprovechando sofisticados algoritmos, buscan fotografías similares entre diferentes servidores.
- En el audio también se detecta el plagio, algunas de las herramientas informáticas que se pueden usar son Shazam y SoundHound (23).

1.4 Algoritmos para la detección de plagio en código fuente

Los algoritmos generalmente utilizados en la detección de plagio en código fuente van desde criterios muy sencillos basados en métricas de software, como el conteo de variables, hasta complejas funciones de codificación y comparación de porciones de texto. Se han logrado identificar tres tendencias fundamentales encabezadas por el uso de los algoritmos *Running Karp-Rabbin* (25), *Greedy String Tiling* (26) y técnicas basadas en funciones *hash* (27) para generar claves que representan de manera casi unívoca a un documento, texto o archivo (16). A continuación se detalla la base lógica y principales características de estos algoritmos.

1.4.1 Algoritmo *Running Karp-Rabin*

La base del algoritmo *Running Karp-Rabin* es la función *Karp-Rabin* utilizada para crear claves en porciones no comparadas del código fuente, las coincidencias son buscadas sobre dicha clave y el tiempo de ejecución

es optimizado con el uso de tablas para el almacenamiento. Contrario a realizar una búsqueda exhaustiva de coincidencias, se puede obtener de la tabla las posiciones iniciales de todas las porciones de código de longitud **S**, procesadas en orden descendente para intentar encontrar primero las coincidencias de mayor longitud que tengan el mismo valor asignado por la función *Karp-Rabin*. Luego de encontrar una coincidencia de la porción de código y la clave a ella asignada, se continúa buscando nuevos elementos hasta que el algoritmo se detiene debido a una diferencia o el fin de la entrada (25).

1.4.2 Algoritmo *Greedy String Tiling*

El algoritmo *Greedy String Tiling* encuentra bloques de código similares comenzando por valores extremos y disminuyendo la longitud hasta un mínimo elegido. En cada ciclo de búsqueda se analizan las porciones de tamaño **S** que coincidan en ambos códigos, las mismas son añadidas a la lista de coincidencias en una estructura del tipo: **posición fuente, posición destino, longitud**; reflejando una similitud encontrada comenzando en la **posición fuente** del primer código y en la **posición destino** del segundo, con una **longitud** determinada por el tercer parámetro. La porción de código es marcada para no tenerla en cuenta en futuras iteraciones y se continúa el proceso con valores **S** menores hasta llegar al umbral inferior de la detección (26).

1.4.3 Algoritmos basados en funciones *Hash*.

El termino *hash* o picadillo es asociado a las funciones *hash* y las tablas *hash* indistintamente. En resumen, es la codificación de un texto en un valor acotado, mediante una función matemática. El objetivo principal de este procedimiento es aprovechar las capacidades de direccionamiento directo de estructuras como los arreglos, ya que, gracias a la conversión de un texto en su correspondiente clave numérica, esta puede ser utilizada como índice del arreglo y realizar operaciones de búsqueda en orden constante; en este caso la complejidad algorítmica está determinada por el rendimiento de la función utilizada para generar la clave *hash* correspondiente.

En la detección de plagio en código fuente, el uso de funciones para convertir el código en una clave numérica, tiene como principal objetivo ganar en velocidad de comparación, debido a que el proceso de obtención de los elementos es constante y óptimo, como antes se señalaba. El funcionamiento, apartando la complejidad de la función *hash*, es bastante simple y sin alejarse mucho de la comparación realizada por otros algoritmos para la detección de plagio, pero con una complejidad computacional menor (26).

1.5 Herramientas para la detección de plagio en código fuente

Existe un considerable número de herramientas para la detección de plagio en código fuente bajo distintos tipos de licencia disponibles en Internet, ya sea en forma de servicio web o como una aplicación descargable, incluso por medio de correo electrónico. Dichas aplicaciones tienen sus más antiguos precedentes en un artículo de Karl J. Ottenstein (27). Se destacan entre ellas el YAP3 de Michael Wise, el MOSS de Alex Aiken y el JPlag de Guido Malpohl. Ellas representan tendencias diferentes en la forma de abordar el problema de la detección de plagio. A continuación se detallan algunas de sus principales características.

1.5.1 YAP3

YAP3 por sus siglas en inglés de ***Y**et **A**nother **P**lague*, es un programa desarrollado en el año 1996, que basa su funcionamiento en utilidades de la consola Unix, fue programado en Perl y los *scripts* pueden ser bajados desde la web del autor (28). Es una herramienta gratuita para fines no comerciales, disponible para los lenguajes C, Pascal y Lisp. Se basa en una mezcla de *Running Karp-Rabin* con *Greedy String Tiling*. Comienza su procesamiento del código eliminando los comentarios y constantes de cadena, convierte todos los textos a letras minúsculas por lo que no tiene en cuenta esta característica. Mapeando los sinónimos a una palabra común simplifica el engaño basado en remplazos de este tipo. Su estrategia para evadir los reordenamientos del código se fundamenta en procesar las funciones de acuerdo a su orden de llamada.

Sus principales desventajas vienen dadas por la rústica representación de los resultados, no puede ser usado en una herramienta que se vaya a comercializar y está disponible para pocos lenguajes de programación.

1.5.2 MOSS

MOSS de sus siglas en inglés para ***M**easure **O**f **S**oftware **S**imilarity* o traducido “medición de similitud de programas”. Es un servicio de Internet desarrollado en el año 1994. Funciona sometiendo a su consideración un grupo de ficheros y el sistema responde con un conjunto de páginas HTML detallando las similitudes detectadas. Es una herramienta gratuita, para su utilización es necesario obtener una cuenta vía correo electrónico del propietario; su uso está limitado a fines no comerciales. El MOSS se encuentra disponible para los lenguajes de programación: C, C++, Java, C#, Python, Visual Basic, JavaScript, FORTRAN, ML,

Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, MIPS assembly, HCL2.

Basa su funcionamiento en la comparación de huellas digitales sacadas de la estructura de los documentos, o códigos fuentes en este caso, luego divide los conjuntos de huellas en porciones basándose en algoritmos n-gramas[8], a las cuales se adjunta información estructural como el número de la página referenciada. Inicialmente el algoritmo crea un índice a distintos lugares dentro de los documentos, identificados con su respectiva huella, para luego crear de todas las coincidencias la huella distintiva del documento. Como último paso las coincidencias son ordenadas de mayor a menor grado, según su nivel de relevancia (27).

MOSS es un poderoso sistema del cual se pueden señalar como características positivas la cantidad de lenguajes de programación que soporta y la calidad de sus resultados. En sentido negativo, la imposibilidad de utilizarlo comercialmente sería una barrera en la aplicación, en el futuro, de algún modelo de negocio en el contexto en que se utilice. Además, su disponibilidad solo como servicio web y no como una aplicación descargable, que pueda ejecutarse en un contexto privado, disminuye su eficiencia en el procesamiento de una cantidad tan grande de códigos fuentes como la presente en jueces en línea.

1.5.3 JPlag

JPlag viene de una mezcla entre los prefijos **J** de Java y **Plag** de *Plagiarism*. Es un sistema para la detección de plagio en conjuntos de códigos fuentes que se encuentra en forma de servicio web desde el año 1996 y su uso es gratuito aunque limitado por la creación de una cuenta. Esta aplicación programada en Java toma como entrada un conjunto de programas y para cada par computa las regiones similares; como resultado genera un reporte en forma de página HTML. JPlag se basa en un algoritmo similar al usado por YAP3 pero con algunas optimizaciones para mejorar su eficiencia. Trabaja convirtiendo el programa de alguno de los lenguajes que acepta: Java, C#, C y C++, en símbolos básicos o *tokens* para luego tratar de completar su estructura con subcadenas de estos símbolos correspondientes al programa con el que se está comparando. El porciento de plagio es deducido con respecto a la cantidad de subcadenas que puedan completarse.

[8] La **n-grama** es una representación redundante de texto que consiste en fragmentos solapados, ya sea a nivel de carácter o de palabra, cuya longitud es **n**. Por ejemplo, las 3-gramas, a nivel de caracteres de “ejemplo” son [eje, jem, emp, mpl, plo]; y las 2-gramas a nivel palabra de “este es solo un ejemplo” son [este es, es solo, solo un, un ejemplo] (16).

Su principal desventaja es que la cuenta necesaria para su uso está sujeta a la aceptación del propietario. Además al estar disponible solo para cuatro lenguajes de programación y los jueces en línea, normalmente, permitir muchos más; ofrece a los usuarios una vía de escape traduciendo a un lenguaje no contemplado en el detector de plagio.

Luego de analizar los principales algoritmos para la detección de plagio y los sistemas existentes, se concluye que no es posible contar con la experiencia de proyectos similares para afrontar la presente problemática. No es conveniente el uso de alguno de los sistemas para la detección existentes, debido a que sus ventajas no se complementan y sus desventajas vienen dadas por factores como condiciones para otorgar la licencia, deficiencia en la cantidad de lenguajes de programación contemplados y principalmente la calidad de los dictámenes al tenerse solo en cuenta la estructura del código fuente. Estos sistemas son incapaces de valorar la repercusión que tienen sobre las similitudes detectadas, la pertenencia del problema a una categoría determinada. Es lógico que el código de diferentes usuarios usado para implementar un algoritmo de camino mínimo, por ejemplo el *Dijkstra*, sea similar ya que el mismo puede encontrarse en varias bibliografías y es considerado un algoritmo clásico (29). Se valoró el funcionamiento y ventajas de las principales tendencias algorítmicas para la detección de plagio, este estudio permitió concluir que el algoritmo *Greedy String Tiling* es el de más común implementación en herramientas detectoras, debido a la baja complejidad en su implementación y la calidad de sus dictámenes.

1.6 Jueces en línea de programación

Un juez en línea de programación es una aplicación web destinada a evaluar, de forma automática, propuestas de solución a un conjunto permanente de problemas o en el contexto de una competencia de programación. Adicionalmente se puede tomar como factor para determinar el potencial de un juez en línea, la cantidad de lenguajes de programación que soporta para evaluar las soluciones, la cantidad de problemas, las modalidades de competencia; además de los clásicos patrones de usabilidad, confiabilidad y disponibilidad.

Hay decenas de jueces en línea alrededor del mundo destacándose el UVA creado en 1995 por Miguel Ángel Revilla y publicado por la Universidad de Valladolid en abril de 1997. Este Juez ha recibido más de 10000000 de códigos fuentes para evaluarlos (3); se destaca también como juez oficial de la ACM-ICPC. El *Sphere Online Judge* (SPOJ) de la Universidad de Gdansk en Polonia, una plataforma capaz de evaluar

soluciones en más de 50 lenguajes, con más de 21800 problemas y disponible en varios idiomas; alrededor de 37968 usuarios envían cerca de 100000 soluciones mensuales (30).

Además del PKU (POJ) de la Universidad de Peking en China se debe mencionar a Codeforces, publicado en Internet desde el 2010 y que se ha convertido en el primer juez en línea de la Web 2.0, desarrollado por Mike Mirzayanov, su principal característica es la constancia con que ha venido celebrando competencias casi semanales.

La UCI comienza a incursionar en el desarrollo de jueces en línea desde el año 2006 y hasta la actualidad se han publicado más de cinco aplicaciones de este tipo, tomaron mayor relevancia el juez en línea de la iniciativa *Xtreme* y la Cátedra de Programación Avanzada (CPAV). Con la experiencia obtenida con estos proyectos surge, en el marco de la inserción de la UCI en el movimiento ACM-ICPC y como vanguardia en Cuba y el Caribe, la iniciativa de publicar un juez para la región; surge así el 5 de junio del 2010 el Juez en Línea Caribeño (COJ).

1.7 Plagio en los jueces en línea

El plagio también está presente en los jueces en línea y es una de las dificultades que presentan actualmente. Se han identificado usuarios que envían soluciones de otros, dándolas como suyas. El 9 de septiembre del 2009 el CPAV detectó mediante una aplicación para la comparación de texto 948 códigos casi idénticos correspondientes a 374 usuarios, en una segunda corrida de la aplicación el 31 de mayo del 2010 fueron 271 posibles casos y 211 los usuarios involucrados (6). La necesidad de detectar el plagio se ha tornado primordial para garantizar la fiabilidad de los resultados en las competencias y las estadísticas; además esta práctica antiacadémica y antiética atenta contra el aprovechamiento que pueden hacer los estudiantes de una aplicación tan útil para las asignaturas de programación.

1.7.1 Herramientas para la detección de plagio en jueces en línea

Se llevó a cabo una búsqueda de herramientas para la detección de plagio usadas por los jueces en línea. Mediante la investigación y comunicación vía correo electrónico con administradores de 16 jueces en línea alrededor del mundo, solo se ha confirmado que el SPOJ y un módulo Académico del juez en línea de la Universidad Regional Integrada (URI) en Brasil, intentaron usar una herramienta de este tipo; limitándose a consumir los servicios prestados por MOSS.

No se ha logrado identificar una tendencia a vincular aplicaciones para detectar plagio en los jueces en línea. En adición, se hizo una búsqueda en Google Scholar, combinándose de varias formas las palabras juez en línea y detección de plagio, no encontrándose en ningún caso algún trabajo representativo además de los publicados como resultados preliminares de esta investigación.

Solo se logró identificar el COJ como juez en línea completamente determinado a detectar plagio. Los jueces de esta aplicación utilizan con este objetivo un módulo para la detección de plagio desarrollado en el año 2012 expresamente con esta función. Este módulo está basado en una mezcla de comparación del código fuente mediante el algoritmo *Greedy String Tiling,* análisis de las características del problema y del perfil del usuario. Los resultados de este módulo fueron favorables y fue la primera solución para la detección de plagio desarrollada específicamente para un juez en línea. No obstante, los tiempos de ejecución son altos e impiden que su uso se generalice más allá de la confirmación de una sospecha entre algunos cientos de soluciones; por ejemplo para detectar plagio en una competencia (10).

1.8 Computación distribuida

El algoritmo *Greedy String Tiling* puede ser encontrado en varias bibliografías por ejemplo en (26). La versión secuencial del algoritmo con una complejidad temporal $O(n^3)$ emplea mucho tiempo en la ejecución de una detección de plagio entre dos códigos fuentes (26). Añadiendo a esta complejidad el procesamiento completo de **m** soluciones comparándose entre ellas la complejidad sería $O(m^2n^3)$ todo ejecutándose en un solo procesador. La distribución de este procesamiento posibilitaría la ejecución de detecciones a gran escala, teniendo en cuenta que en apenas unas horas en el marco de la celebración de una competencia, un juez en línea tiene que evaluar cientos o miles de soluciones (31).

Una solución distribuida es una “solución informática con componentes hardware y software, que están en ordenadores conectados en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor” (11). El protocolo AMQP tiene como objetivo coordinar la comunicación mediante mensajes a través de una cola de mensajería (32).

1.8.1 AMQP

En el año 2004 JPMorga Chase comenzó el desarrollo del protocolo Avanzado de Colas de Mensajería (AMQP, por sus siglas en inglés) con la corporación iMatix. AMQP fue diseñado desde un principio como un

estándar abierto que resolvería la mayoría de las necesidades de las colas de mensajería. Debido a que es un estándar abierto, cualquiera puede implementarlo y cualquiera que lo implemente pasa automáticamente a ser interoperable con cualquier servidor de colas de mensajería (MQ) que también lo implemente. AMQP promete una conexión dinámica de información en tiempo real desde cualquier productor a cualquier consumidor sobre un bus de software (32).

AMQP comprende tanto una red de protocolo, que especifica lo que el cliente de aplicaciones y servidores de mensajes deben enviar a través del medio de conexión para interoperar entre sí, y un modelo de protocolo, que especifica la semántica que una aplicación AMQP debe obedecer para interoperar con otras implementaciones.

Estos sistemas típicamente evocan imágenes de un servidor centralizado, monolítico, primero en entrar primero en salir (FIFO); sistemas que aceptan colas de mensajes en un extremo y dispensación de ellos desde el otro. AMQP toma un enfoque más modular, dividiendo el mensaje entre las diferentes colas de mensajes a través de *exchanges*[9] (34).

AMQP conecta a través de (34):

- Organizaciones - aplicaciones en diferentes organizaciones.
- Tecnologías - aplicaciones en diferentes plataformas.
- Tiempo - aplicaciones que no tienen que estar disponibles de forma simultánea.
- Espacio - funciona de forma fiable a distancia, a través de redes de bajo rendimiento.

AMQP fue diseñado con las siguientes características (34):

- Seguridad
- Confiabilidad
- Interoperabilidad
- Estándar
- Abierto

En la actualidad hay varias implementaciones de agentes de AMQP, se pueden citar las siguientes:

[9] Los ***exchanges*** son elementos de la arquitectura del protocolo AMQP que se encargan de aceptar mensajes de una aplicación productora y enrutarlos a una cola de mensajes.

- SwiftMQ, un servicio comercial de mensajes para java (JMS).
- Windows Azure *Service Bus*, servicio de mensajería de Microsoft basado en la nube.
- Apache Qpid, un proyecto de código abierto en la fundación Apache.
- Apache ActiveMQ, un proyecto de código abierto en la fundación Apache.
- Apache Apollo, una versión modificada de código abierto del proyecto ActiveMQ.
- RabbitMQ, un proyecto de código abierto, soporta AMQP 1.0 y otros protocolos mediante *plugins*.

RabbitMQ es un sistema de mensajería para aplicaciones, robusto y fácil de usar. Es uno de los más prometedores y el de más creciente interés en la comunidad de desarrolladores (35). Está disponible para la mayoría de los sistemas operativos más usados y soporta un número significativo de plataformas de desarrollo. Tiene versiones de código abierto y comercial. Es, comparado con las otras implementaciones de AMQP, la mejor atendiendo al rendimiento y funcionalidades (36).

1.8.2 RabbitMQ

RabbitMQ es un software de negociación de mensajes de código abierto y entra dentro de la categoría de *middleware* de mensajería. Implementa el estándar *Advanced Message Queuing Protocol* (AMQP). El servidor RabbitMQ está escrito en Erlang[10] y utiliza el *framework Open Telecom Platform* (OTP) para construir sus capacidades de ejecución distribuida y conmutación ante errores. *Rabbit Technologies Ltd*, la compañía que lo desarrolla, fue adquirida en abril de 2010 por la división SpringSource de VMWare (37). A partir de este momento, es esta última compañía la que desarrolla y da soporte para RabbitMQ. El código fuente está liberado bajo la licencia *Mozilla Public License*.

El proyecto RabbitMQ consta de diferentes partes:

- El servidor de intercambio RabbitMQ.
- Pasarelas para los protocolos HTTP, XMPP y STOMP.
- Librerías de clientes para Java y el *framework* .NET. (Librerías similares para otros lenguajes se encuentran disponibles por parte de otros proveedores).

[10] Parte importante del éxito de Rabbitmq se debe a la elección de **Erlang** como lenguaje de programación, debido a que maneja la concurrencia y la paralelización de forma explícita y eficiente.

- El *plugin Shovel* o pala es el que se encarga de replicar mensajes desde un corredor de mensajes a otros.

1.9 Sistemas de caché distribuida

De acuerdo al análisis de las soluciones y del comportamiento de los usuarios durante la solución de problemas, en un juez en línea se puede detectar plagio entre:

- Dos soluciones.
- Soluciones de usuarios específicos.
- Soluciones de un problema específico.
- Varias soluciones entre ellas.
- Una solución contra otras.

Se puede observar que en todos los casos hay acceso redundante a la misma solución o los datos de un mismo usuario o problema. Actualmente el acceso a la base de datos relacional empleada es el punto crítico que atenta contra el funcionamiento del sistema, que no utiliza ninguna solución para cachear los contenidos luego del primer acceso.

Un sistema de caché distribuida de alto rendimiento que almacene los códigos fuentes luego del primer acceso permitirá aliviar la carga soportada por la base de datos relacional y tendrá una repercusión positiva en el rendimiento general de los detectores.

Entre los sistema de este tipo más utilizados se encuentran Memcached y Redis. Memcached es un sistema libre, de código abierto, de alto rendimiento, de caché en memoria distribuida, genérico en naturaleza, pero con la intención de uso en la optimización de aplicaciones web dinámicas aliviando la carga de bases de datos (38).

Redis es un avanzado almacén o caché clave-valor. Usualmente referenciado como un servidor estructurado dado que las claves pueden contener cadenas, hashes, listas, conjuntos, conjuntos ordenados, mapas de bits, e "*hyperloglogs".* Este modelo tiene todo el conjunto de datos en memoria y a veces o cuando el número de registros cambiados llega a cierto valor, estos son escritos asincrónicamente en disco. Podrían

perderse un pequeño número de valores que es aceptable[11] para la mayoría de las aplicaciones pero es tan rápido como una base de datos en memoria. (40)

Aunque ambos sistemas son similares, en la literatura Redis se referencia como más completo, con muchas más funcionalidades e incluso mejor rendimiento en determinados contextos de ejecución (40).

1.9.1 Redis

Las técnicas y herramientas utilizadas para el almacenamiento y consulta de datos han crecido a un ritmo increíble. El término NoSQL apareció por primera vez en los 90, pero el concepto no se introdujo realmente de manera seria en el mundo de las bases de datos hasta el 2009. El surgimiento de las alternativas de bases de datos NoSQL refleja la necesidad que tiene la industria del software de disponer o de evaluar rápidamente volúmenes crecientes de datos.

Redis es un modelo de datos NoSQL que comenzó a desarrollarse en marzo del 2009 por Salvatore San Filippo, para mejorar los tiempos de respuesta de un producto llamado LLOGG. Fue ganando popularidad, hasta que en marzo del 2012 la empresa VMWare contrató a Salvatore para trabajar a tiempo completo en Redis. Poco después contrató también a otros de los principales desarrolladores de Redis, entre ellos a Pieter Noordhuis (41).

1.9.2 Rendimiento

Una de las características de Redis es su velocidad, para medirlo incluye una herramienta llamada *redis-benchmark*. Con esta herramienta se puede ver el rendimiento que alcanza en sus operaciones. En escenarios de datos no durables, solo usando memoria RAM, el rendimiento puede ser extremo comparado con motores de bases de datos relacionales, tampoco hay una notable diferencia entre lectura y escritura de datos. Además de todo esto, Redis se destaca por ser una base de datos con un rendimiento muy elevado, se define como una base de datos en memoria con persistencia para datos, que puede ser desactivada.

[11] En el contexto en que se pretende aplicar esta tecnología se tendrá en cuenta esta pérdida de datos.

1.9.3 Replicación

Redis soporta la replicación del tipo maestro-esclavo, pudiéndose replicar los datos de un servidor a muchos esclavos, también un esclavo puede ser maestro para otro esclavo, lo que permite soportar en Redis una replicación en forma de árbol. Los esclavos permiten la escritura de datos, lo que puede ocasionar inconsistencias en los datos no intencionales. (41)

La función de publicación y subscripción está totalmente soportada, cuando un cliente esclavo se subscribe a un canal este recibe un *feed* completo de publicaciones del maestro, replicando así en todo el árbol. La replicación es útil para escalar la lectura y redundar los datos. (41)

La replicación de Redis es muy sencilla de utilizar y configurar, la replicación maestro-esclavo permite que los servidores esclavos sean copias exactas de los servidores maestros. A continuación se exponen algunos elementos importantes acerca de la replicación en Redis (41):

- Usa replicación asíncrona.
- Un maestro puede tener varios esclavos.
- Los esclavos son capaces de aceptar conexiones de otros esclavos. Aparte de la conexión de un número de esclavos al mismo maestro, los esclavos también se pueden conectar a otros esclavos; formándose una estructura jerárquica donde pueden haber varios maestros.
- La replicación es sin bloqueo en el lado del maestro. Esto significa que el maestro continuará manejando las consultas cuando uno o más esclavos llevan a cabo la sincronización inicial.
- La replicación es también sin bloqueo en el lado esclavo. Mientras que el esclavo está realizando la sincronización inicial, puede manejar consultas utilizando la versión antigua del conjunto de datos, suponiendo que Redis se haya configurado para hacerlo.
- Es posible usar la replicación para evitar el costo de escritura cuando el maestro escribe el conjunto de datos en el disco, solo habría que configurarlo en el maestro para evitar ahorro, luego conectar un esclavo configurado para salvar periódicamente.
- Los esclavos son capaces de volver a conectarse automáticamente cuando el enlace maestro-esclavo deja de funcionar por alguna razón. Si el maestro recibe múltiples peticiones de sincronización esclava concurrentes, realiza un único fondo de ahorro con el fin de servir a todos ellos.

1.10 Conclusiones parciales

En este capítulo se profundizó en los conceptos relacionados con el dominio de los sistemas para la detección de plagio. Se concluye que:

- No se logró identificar una solución para la detección de plagio que pueda aplicarse en jueces en línea y tenga tiempos de ejecución favorables.
- AMQP es una opción factible para el intercambio de mensajes entre procesos distribuidos en una red de computadoras.
- RabbitMQ implementa eficientemente el protocolo AMQP.
- En un contexto de detección distribuida el acceso a la base de datos relacional se convertiría en un punto crítico que afectaría el rendimiento.
- Un sistema de caché distribuida de alta velocidad puede disminuir el uso de las bases de datos relacionales en la detección distribuida de plagio en jueces en línea.
- Redis es un sistema de caché distribuida con características adicionales y un rendimiento sobresaliente.

Propuesta de solución

La función del presente capítulo es documentar el proceso de desarrollo de una solución para la detección de plagio en jueces en línea de programación. Se propone un detector distribuido apoyado en un sistema de caché también distribuido, que permitirá detectar plagio en jueces en línea con menores tiempos de ejecución que las herramientas detectoras identificadas. El capítulo termina con la selección de los algoritmos para la detección de plagio.

2.1 Esquema de solución

El esquema de solución que se propone parte de la diferenciación entre una detección simple y múltiple. La razón de que se establezca esta diferencia está dada por el procesamiento que se puede hacer con ambas. Una detección múltiple está destinada a valorar los porcientos de similitud entre los códigos, para detectar soluciones sospechosas. Una detección simple[12] se realiza con el objetivo de confirmar una sospecha a partir del análisis de información adicional, como las coincidencias encontradas.

La solución que se propone, será desacoplada por completo del código del juez en línea y empaquetada en un fichero que tendrá dos contextos de ejecución completamente diferentes. El primero será utilizado para la detección simple y formará parte del juez en línea, facilitando una API para la detección de plagio y un conjunto de utilidades necesarias para conectar con el segundo contexto: el distribuido.

Una detección múltiple será ejecutada en un contexto distribuido donde el sistema procesará las soluciones en varias computadoras que tengan acceso a un sistema de mensajería, mediante el cual se comunicarán con el juez en línea escuchando órdenes de detección y devolviendo las respectivas respuestas.

[12] Una **detección simple** solo implica a una pareja de códigos fuentes.

2.1.1 Detección simple

En el caso de la detección simple se propone realizarla en el propio juez en línea utilizando para esto la API brindada por el detector. A través de un único punto de conexión entre el Juez y el sistema detector, se debe establecer una conexión con las funcionalidades brindadas a través de dos interfaces: “Detector de plagio” con las funciones básicas para ordenar una detección, establecer los parámetros necesarios para esta y “Dictámenes” para encapsular el resultado.

La propuesta general de funcionamiento del detector de plagio se abstrae de la forma de detección mediante el uso de la interfaz “Detector”, que puede tener implementaciones variadas entre las que no pueden faltar la dedicada a la comparación del código fuente y a las características (clasificación) del problema. Otras posibles son: perfil del usuario, métricas de software, comportamiento en tiempo de ejecución y estilo de codificación.

Los resultados independientes de los detectores se mezclan con una determinada ponderación establecida a apreciación directa de los expertos o mediante aprendizaje automático, por ejemplo con una red bayesiana tomando como base de conocimiento el historial de detecciones y las opiniones de los expertos. El esquema propuesto se puede observar en la Figura 1.

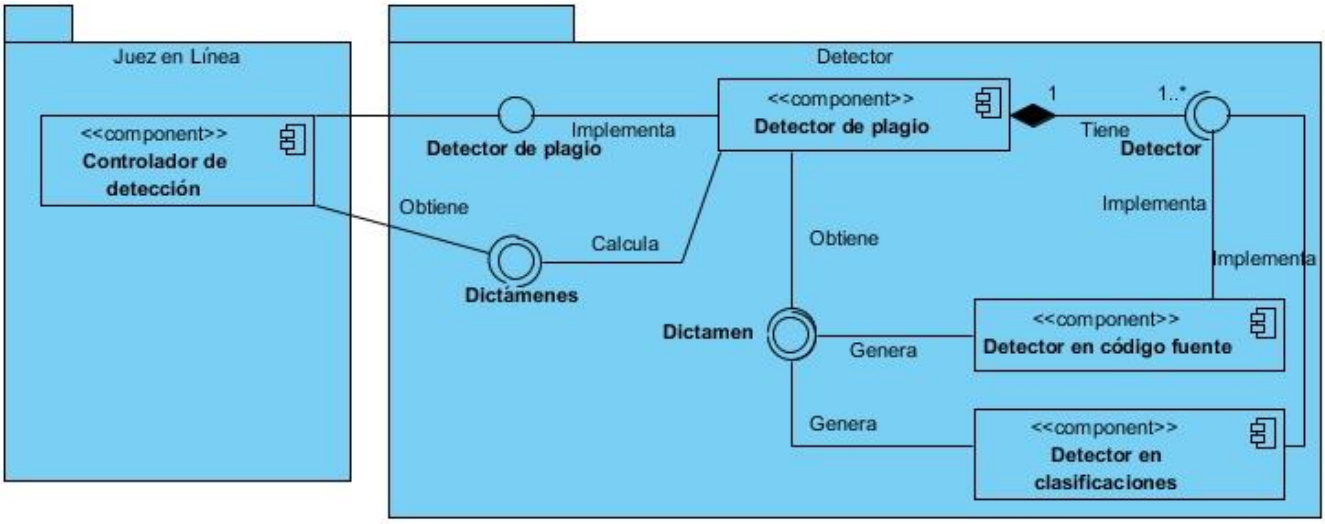


Figura 1: Esquema propuesto para la detección simple de plagio.

2.1.2 Detección múltiple

Los componentes principales que diferencian la detección múltiple de la simple son el servidor de mensajería y el componente de conexión, a través del cual el juez en línea enviará mensajes del tipo (solución a, solución b) que serán almacenados en una cola según su nivel de prioridad y entregados a los detectores distribuidos en la red siguiendo una estrategia *Round-Robin*[13]. La respuesta es entregada al juez en línea también a través de una cola.

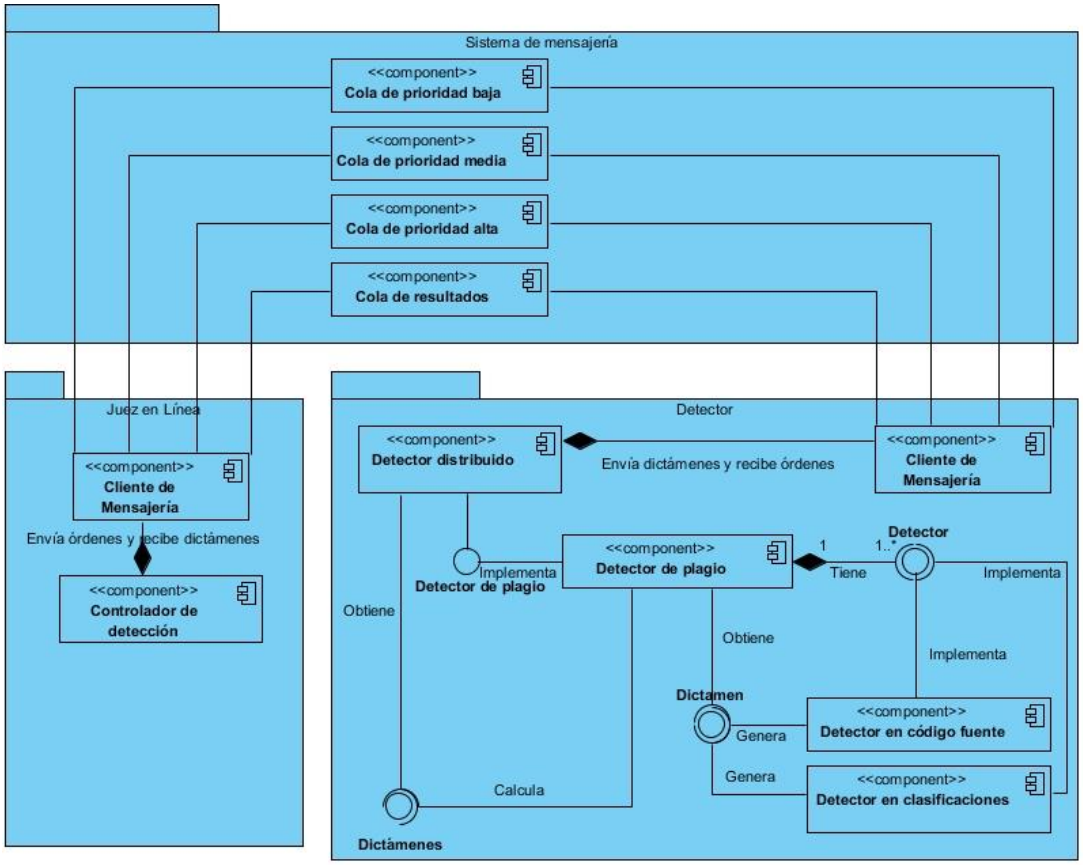


Figura 2: Esquema propuesto para la detección múltiple de plagio.

[13] **Round-Robin** es un método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento (59).

Como se puede observar en la Figura 2 existe un desacoplamiento total entre el juez y los detectores; estos están separados a través del espacio físico y también del tiempo, pues las colas sirven de contenedor intermedio. Con este esquema se garantiza la detección, aun cuando al momento de ordenada no existan detectores disponibles. Los demás componentes se mantienen de la detección simple, pero esta vez como un proceso independiente y no como parte del juez en línea.

2.1.3 Sistema de caché distribuida

En un juez en línea se puede detectar plagio entre:

- Dos soluciones.
- Soluciones de usuarios específicos.
- Soluciones de un problema específico.
- Varias soluciones entre ellas.
- Una solución contra otras.

Se puede observar que en todos los casos hay acceso redundante a la misma solución o los datos de un mismo usuario o problema siendo actualmente el acceso a la base de datos relacional empleada, el punto crítico que atenta contra el funcionamiento del sistema que no utiliza ninguna solución para cachear los contenidos luego del primer acceso.

Una vez distribuida la detección de plagio, la base de datos relacional es el único punto en que la detección sigue afectando los recursos computacionales del Juez en Línea. Es en la base de datos donde todos los detectores tienen que buscar el código fuente que van a analizar. En una detección de plagio entre n soluciones se necesitan acceder $2n^2$ veces a los códigos fuentes, suponiendo que no se efectuara ningún filtro, como por ejemplo no detectar plagio entre una solución y ella misma.

Los $2n^2$ códigos fuentes necesarios están dados por la detección todos contra todos de n soluciones. En cada una de estas n^2 detecciones se utilizan 2 códigos fuentes; de ahí el planteamiento.

Con el objetivo de disminuir el acceso a la base de datos relacional, se propone utilizar un sistema de caché distribuida de alto rendimiento que almacene los códigos fuentes luego del primer acceso. El uso de esta solución permitirá que solo n de las $2n^2$ peticiones las responda la base de datos relacional encargada de almacenar toda la información utilizada por el Juez en Línea.

2.1.4 Esquema general para la detección de plagio

El esquema completo contempla algunos componentes adicionales como el sistema de cacheo, los analizadores léxicos y las vistas. Estos no se tuvieron en cuenta para los modelos anteriores porque no son necesarios para entenderlo y se prefirió mantenerlos lo más simple posible.

La aplicación de este esquema al desarrollo de un detector de plagio para jueces en línea de programación, permitirá obtener una solución que contará con las siguientes características:

- Completamente desacoplado de los jueces en línea, lo que permitirá ampliar su uso a tantas aplicaciones de este tipo como se desee.
- No se enmarca la solución en un solo criterio para detectar plagio, sino que permitirá extender con facilidad los parámetros que se tienen en cuenta en el dictamen.
- Independencia del sistema de mensajería, dado que la única característica requerida es la más básica posible: envío y recepción de mensajes.
- Independencia del sistema de caché distribuida, dado que la única característica requerida es la más básica posible: insertar y obtener elementos.

Elementos como la utilización de más de una cola para brindar la posibilidad de establecer prioridades en las detecciones, la modelación de seis analizadores léxicos, las vistas que están reflejadas en el esquema o la implementación de exactamente dos detectores; se referencian para aumentar la claridad en la propuesta.

La utilización y número de los elementos antes mencionados depende del contexto en que se aplique la propuesta. La más simple implementación podría estar compuesta por un analizador léxico, una vista para ordenar la detección o recibir el dictamen y solo una cola de mensajería por la cual realizar los intercambios de datos. Los detectores si deben ser mínimo dos: para el código fuente y la clasificación.

El esquema completo que se propone para la detección de plagio en jueces en línea, con los componentes adicionales antes mencionados se puede observar en la Figura 3.

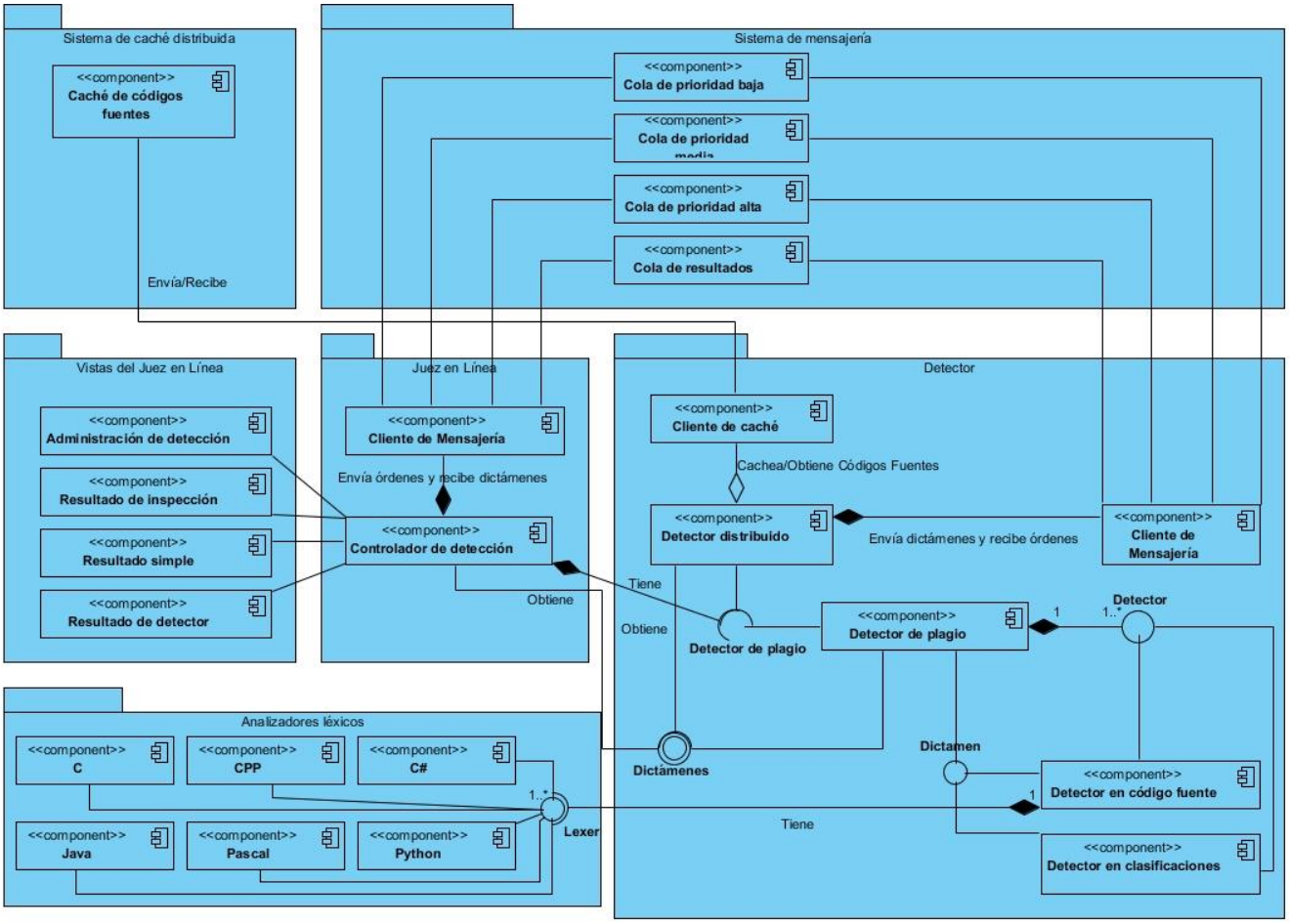


Figura 3: Esquema propuesto para la detección de plagio en jueces en línea de programación.

2.2 Definición de algoritmos para la detección de plagio

La base algorítmica se debe sustentar en varios criterios que al complementarse generen un dictamen. No pueden faltar entre los criterios analizados, un detector para el análisis de la estructura del código fuente y la clasificación del problema. Se recomienda la aplicación del algoritmo *Greedy String Tiling* (26), aunque el esquema contempla la utilización de cualquiera de los documentados en la literatura.

En general el único requerimiento para la implementación de detectores es que reciban dos soluciones y generen un dictamen en forma de probabilidad de plagio (un valor en el rango [0, 1]) y una lista de coincidencias entre las soluciones.

Por ejemplo, la implementación de un detector basado en el algoritmo *Greedy String Tiling* recibirá la solución en la que podrá obtener los códigos fuentes y los lenguajes en que fueron escritos. El retorno de este detector será un valor en el rango [0,1] donde valores cercanos a 1 indicarían alta probabilidad de plagio; además retornará una lista de coincidencias, que en su caso corresponderán a porciones continuas de código fuente que son coincidentes en ambas soluciones.

2.2.1 Detector de plagio basado en la estructura del código fuente

Con respecto al algoritmo *Greedy String Tiling* y el resto de los algoritmos existentes para la detección de plagio en código fuente, pueden ser encontrados en varias bibliografías (26). Este no es el caso del detector basado en la clasificación del problema, para el que se propone una estrategia sobre la base de la experiencia en el campo de acción y criterios no necesariamente complejos. Este detector permite equilibrar la ineficacia de los resultados obtenidos con los sistemas de detección existentes al aplicarlos al entorno de los jueces en línea.

2.2.2 Analizadores léxicos

El algoritmo *Greedy String Tiling* no trabaja directamente sobre el código fuente sino sobre una representación intermedia, un flujo de *tokens* obtenidos de la fase de análisis léxico del proceso de compilación/interpretación de un lenguaje. Para obtener este flujo de *tokens* es necesario programar un analizador léxico para las gramáticas de cada lenguaje que se desea puedan procesarse en busca de plagio. Para facilitar el desarrollo de los analizadores léxicos, el desarrollador puede apoyarse en un generador de analizadores léxicos; que no es más que una herramienta que construye a partir de una gramática, el código fuente correspondiente al analizador léxico que la interpreta. Entre las herramientas de este tipo más reconocidas se encuentran:

- ANTLR.
- JavaCC.
- SableCC.
- GNU Bison.

Cualquiera de ellas puede ser de ayuda en el complejo proceso de implementar los analizadores léxicos, se recomienda ANTLR pues permite generar los analizadores en varios lenguajes y en su web oficial se encuentran públicas gramáticas para casi todos los lenguajes de uso común.

Las búsquedas en Twitter usan ANTLR para parsear 2 billones de consultas diarias. Los lenguajes para Hive y Pig, el almacén y sistema de análisis de Hadoop, ambos usan ANTLR. Lex usa ANTLR para extraer información de textos legales. Oracle usa ANTLR dentro del Entorno de Desarrollo Integrado (IDE) "SQL Developer". NetBeans parsea C++ con ANTLR. El lenguaje HQL en el ORM Hibernate fue construido con ANTLR (42).

2.2.3 Dictamen basado en la estructura del código fuente

El dictamen final queda formulado, de acuerdo a las parejas de fragmentos de códigos detectados, según la Ecuación 1. Se establece que la similitud entre dos códigos está dada por la razón entre "la suma de las longitudes de los fragmentos similares o coincidencias, multiplicado por dos debido a que están presentes en ambos códigos"; entre la longitud total (26).

Tabla 3: Leyenda para la Ecuación 1.

Variable	Significado
A	Solución A.
B	Solución B.
S	Conjunto de similitudes detectadas entre la solución A y B.
l	Longitud de la similitud.
c	Código fuente de la solución.

$$similitud(A,B) = \frac{2(\sum_{x \in S} x_l)}{|A_c| + |B_c|}$$

Ecuación 1: Similitud de acuerdo a las coincidencias detectadas en el código fuente.

Cómo se puede observar en la ecuación el valor de similitud está acotado en el rango [0, 1], dado que el denominador es fijo, el resultado está dado por la suma de la longitud de los fragmentos similares. En caso de no existir fragmentos similares el numerador es cero y el resultado por consiguiente también. Por el contrario, en caso de que los códigos sean idénticos, el resultado es uno.

2.2.4 Detector de plagio basado en la clasificación del problema

Para el desarrollo del detector basado en la clasificación del problema, se debe comenzar con la creación de un sistema de clasificación en el juez en línea (Anexo 1) capaz de definir para cada problema, a cuál área del conocimiento pertenece y a partir de esta información se podrá inferir los posibles algoritmos que lo solucionan.

Si resolver un problema implica encontrar la distancia mínima de un nodo a los demás, es probable encontrar entre sus clasificaciones: “búsqueda de caminos mínimos”. Además, si es necesarios realizar algún tipo de procesamiento, por ejemplo ir y volver a la mayor cantidad de nodos contando con un límite de distancia total a recorrer, lo cual podría resolverse con una estrategia golosa; entonces esta clasificación también debe ser incluida.

En consecuencia se analizaría un problema en el cual sus soluciones tendrían una porción de código correspondiente a un algoritmo de caminos mínimos, típicamente Dijkstra[14], y otra porción de código con un ordenamiento y ciclo de procesamiento, típicos en la aplicación de estrategias golosas. Ambos bloques de código son prácticamente irrelevantes en la detección de plagio; pues el algoritmo Dijkstra salvo detalles, siempre será el mismo. Igual sucederá con la estrategia golosa, se parecerán mucho todas las implementaciones.

La comparación de las estructuras de problemas solubles mediante algoritmos clásicos, necesariamente arrojará altos niveles de similitud y provocará la ocurrencia de un número elevado de falsos positivos. Esta afirmación tiene en cuenta que en los jueces en línea es muy común encontrar problemas de este tipo. Desde este punto de vista se considera ad-hoc como la clasificación prototipo para cualquier detección.

Ad-hoc es una frase proveniente del latín que significa “para esto”, en el ámbito informático es usada con frecuencia para referenciar algoritmos cuyo desarrollo no está sujeto a ningún patrón, categoría o que tenga similitud con algo conocido y por consiguiente cuyo resultado no es generalizable.

[14] **Dijkstra** es un algoritmo para determinar el camino más corto, en un grafo con pesos, dado un vértice origen al resto de vértices. Su nombre se refiere al autor Edsger Dijkstra.

2.2.5 Dictamen basado en la clasificación del problema

Se debe estimar, según las clasificaciones de todos los problemas del juez en línea y las soluciones de los usuarios, la longitud promedio del código que debería emplearse para implementar determinado algoritmo. Una vez estimado este valor se propone la Ecuación 2 para el cálculo de la probabilidad de plagio teniendo en cuenta la clasificación del problema analizado.

Tabla 4: Leyenda para la Ecuación 2.

Variable	Significado
A	Solución A.
B	Solución B.
c	Código fuente.
s	Solución enviada al Juez en Línea.
f	Conjunto de clasificaciones del problema objetivo de la solución.
l	Longitud estimada.

$$similitud(A,B) = \frac{|A_c| + |B_c| - 2\sum_{F\,\epsilon\,[A_f \cap B_f]} F_l}{|A_c| + |B_c|}$$

Ecuación 2: Similitud de acuerdo a las coincidencias detectadas con el comparador de clasificaciones.

La ecuación resta a la suma de las longitudes del código de ambas soluciones, la cantidad que se espera pertenezca a la implementación de las clasificaciones comunes entre ambos problemas. El resultado es la longitud que se supone haya sido parte de la creatividad del programador y no de la experticia ganada en la implementación de algoritmos de la categoría correspondiente o la aplicación de un algoritmo documentado. La razón entre este valor y la suma de las longitudes del código de ambas soluciones es la probabilidad de que este sea plagio.

Nótese como soluciones a problemas que no tengan clasificaciones coincidentes darían 1.0 de probabilidad de plagio (de acuerdo a este criterio), pues sería cero el término correspondiente a la sumatoria de las longitudes de las clasificaciones coincidentes.

2.2.6 Dictamen general de la detección de plagio en jueces en línea

Para calcular un dictamen general de la detección de plagio en jueces en línea, los resultados se deben unir utilizando un promedio ponderado que es una "medida más exacta para obtener resultados de diversos datos con diferentes pesos o grados de importancia" (43). Esta estrategia es utilizada en otras investigaciones para mediar el promedio ponderado de las calificaciones de los estudiantes universitarios (44), para el cálculo de la importancia de la caza para la alimentación humana, ponderando el consumo diario de alimentos (45) y en economía para el cálculo del costo de capital promedio ponderado (46).

Tabla 5: Leyenda para la Ecuación 3.

Variable	Significado
A	Solución A.
B	Solución B.
D	Conjunto de detectores aplicados a las soluciones A y B.
d	Dictamen de la detección.
p	Ponderación del detector.

$$Dictamen(A,B) = \frac{\sum_{x \in D} x_d * x_p}{\sum_{x \in D} x_p}$$

Ecuación 3: Dictamen general del sistema para la detección de plagio.

El efecto que puede tener el promedio ponderado de múltiples criterios, para la detección eficaz de plagio en jueces en línea de programación, está dado por el correcto ajuste de los pesos de cada detector. Se pudiera implementar alguna estrategia basada en técnicas de inteligencia artificial que permitiría ajustar dinámicamente estos valores mediante aprendizaje automático. El autor ha valorado, por ejemplo, utilizar una red bayesiana tomando como base de conocimientos el historial de detecciones y las opiniones de los expertos.

Aunque la aplicación de inteligencia artificial a este problema queda fuera del marco de esta investigación y quedará reflejado en las recomendaciones; se propone una estrategia simple pero eficaz:

- Establecer siempre ponderaciones mayores de lo necesario, teniendo en cuenta que es preferible, falsos positivos que plagio no identificados.

- Ajustar el peso de cada detector a un punto medio entre su dictamen y la apreciación de los jueces, si el dictamen es mayor que la apreciación.
- Ajustar el peso de cada detector a la apreciación de los jueces si el dictamen es menor que la apreciación.

Con este procedimiento se espera que a medida que se realicen detecciones, los pesos de cada detector tiendan a un punto ideal en que sean capaces de expresar exactamente la importancia que tiene su dictamen en la detección final.

2.2.7 Evaluación de los dictámenes

El dictamen debe ser evaluado por los administradores que laboran como jueces de plagio, los cuales son los encargados de emitir criterios sobre el dictamen final de la Ecuación 3 y los dictámenes de las detecciones independientes (Anexo 6). El objetivo de esos criterios es enriquecer la calidad de los dictámenes en futuras versiones y determinar las acciones a llevar a cabo con los usuarios involucrados en un plagio, detectado por el sistema y ratificado por los jueces.

Los detectores de plagio en jueces en línea deben implementar funcionalidades para la notificación a los usuarios, sanción temporal de denegación de servicios, sanción permanente y pérdida de su cuenta en el sistema.

La evaluación de los jueces es fundamental, dado que ningún sistema para la detección de plagio es capaz de afirmar con total seguridad cuando un código fue programado a partir de otro. Estos sistemas solo pueden emitir un dictamen basado en el grado de similitud, pero son incapaces de determinar la razón por la cual los códigos son parecidos (47).

Con el objetivo de unir los criterios de varios jueces de plagio en un dictamen definitivo se puede profundizar en el campo de la estimación de consenso en criterios de expertos. En (48) se describe una investigación relacionada con las “Técnicas para la representación del conocimiento causal: un estudio de caso en Informática Médica”, utilizando mapas cognitivos difusos para estimar el consenso entre criterios de expertos.

2.3 Conclusiones parciales

Al finalizar el capítulo destinado al desarrollo de la solución se concluye que:

- La propuesta estará dividida en detección simple y múltiple con una solución común utilizada en contextos diferentes.
- La detección múltiple se realizará en un contexto distribuido.
- La aplicación de un sistema de caché distribuida permitirá disminuir el uso de los recursos computacionales del juez en línea y tendrá un repercusión positiva en los tiempos de ejecución.
- La detección se basará en múltiples criterios entre los que se encontrarán, el código fuente y la clasificación del problema.

Implementación y validación

Continuación se documentan los aspectos fundamentales relacionados con la validación de los resultados obtenidos mediante la presente investigación. Se realiza un análisis comparativo basado en un conjunto de pruebas destinadas a contrastar las afirmaciones realizadas con la hipótesis de investigación planteada. Se caracteriza la capacidad de procesamiento mediante experimentos y la satisfacción con la propuesta mediante la aplicación de la técnica de Iadov. El cumplimiento de los requisitos se determina con pruebas funcionales utilizando el método de caja negra y la técnica partición equivalente.

3.1 Implementación en el Juez en Línea Caribeño

Para constatar la validez de la solución propuesta, se implementó en el Juez en Línea Caribeño utilizando las siguientes tecnologías:

- Java, como lenguaje de programación.
- Spring, *como Framework* de desarrollo.
- PostgreSQL, como base de datos relacional.
- Protocolo Avanzado para Colas de Mensajes (AMQP), como protocolo para el intercambio de mensajes entre los procesos distribuidos.
- RabbitMQ, como sistema para el intercambio de mensajes que implementa el protocolo AMQP.
- Spring-AMQP, como librería para la conexión mediante el protocolo AMQP.
- Redis, como sistema de caché distribuida.
- Jedis, como librería para la conexión con Redis.

Los detectores fueron implementados sobre la base de dos criterios: el código fuente y la clasificación del problema. Para el análisis del código fuente se utilizó el algoritmo *Greedy String Tiling*. Para la clasificación del problema se utilizó la propuesta realizada en el epígrafe 0. El cálculo de la longitud esperada por clasificación se implementó en forma de procedimiento almacenado en la base de datos relacional. Se realiza periódicamente un procesamiento estadístico que determina primeramente la longitud promedio de

las clasificaciones, teniendo en cuenta los envíos a problemas con una sola clasificación y luego este valor es utilizado para procesar los envíos a problemas con múltiples clasificaciones.

Los analizadores léxicos necesarios para *parsear* el código fuente y convertirlo en la entrada necesaria para el algoritmo *Greedy String Tiling*, se implementaron con la ayuda de ANTLR para los lenguajes de programación C++, Java, C#, C, Python y Pascal, que son en ese orden los de mayor uso en el COJ representando el 99.2% del total de las soluciones enviadas hasta el momento al Juez en Línea (49). En la Figura 4 se grafican los envíos realizados por lenguaje al COJ.

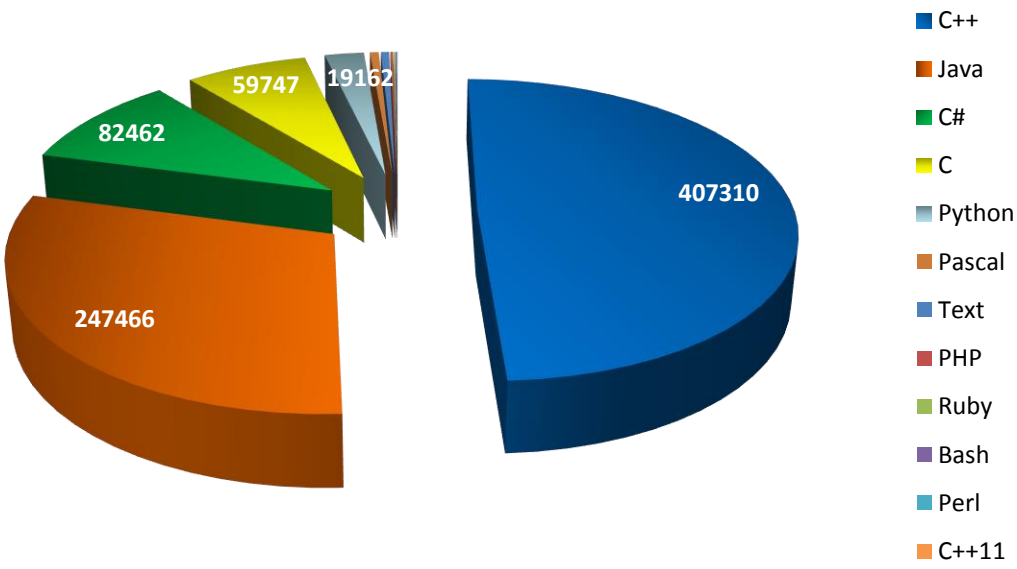


Figura 4: Cantidad de envíos al COJ por lenguaje.

Se implementó una vista administrativa que permite detectar plagio con diferentes proyecciones:

- Envíos a un problema determinado.
- Una pareja determinada de envíos.
- Un rango de envíos.

- Especificar un envío como pivote[15].
- Filtro de lenguaje para comparar solamente soluciones de lenguajes coincidentes.
- Filtro de evaluación para comparar solamente soluciones aceptadas.
- Filtro de equivalencia para no comparar una solución con sí misma.
- Filtro de usuarios para no comparar soluciones correspondientes a un mismo usuario.

Se implementaron dos vistas para los resultados, la primera de ellas en forma matricial enriquecida con una escala de colores desde verde hasta rojo según la probabilidad de plagio, que permite identificar con facilidad los elementos sospechosos, ver Figura 5. La segunda vista es en forma de lista ordenada descendentemente de acuerdo al dictamen. Se implementaron además vistas específicas para cada detector donde se muestran las coincidencias que motivaron su dictamen (Anexos Anexo 7 y Anexo 8).

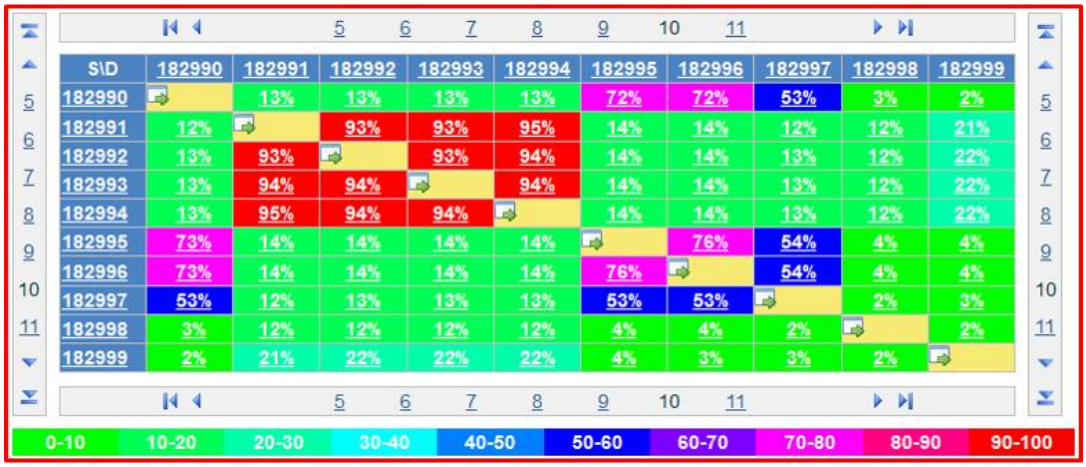


Figura 5: Vista matricial con filtro aplicado de misma solución.

3.2 Pruebas de software

Con el objetivo de determinar si la solución detectora de plagio se considera lista para ser desplegada y en funcionamiento en el COJ, debe someterse previamente a una etapa de pruebas rigurosas, con el fin de analizar si cumple con las funcionalidades requeridas. Las pruebas que se le realizaron a la aplicación son un elemento crítico para la garantía de la calidad, representan además la revisión final de las especificaciones, los requerimientos, el análisis, diseño e implementación. El principal objetivo de estas

[15] Un envío **pivote** es utilizado para comparar una solución contra otras.

pruebas es descubrir errores y corregirlos posteriormente para que la solución final cumpla con lo previsto y tenga la calidad requerida. Aunque las pruebas no pueden asegurar la ausencia de defectos; sí pueden demostrar la existencia de estos (50).

Las pruebas a la aplicación se centrarán en las pruebas de sistema. A continuación se presenta una descripción de las pruebas realizadas al sistema y las estrategias utilizadas para validar la solución desarrollada.

3.3 Pruebas de sistema

Las pruebas de sistema se usan para verificar la aplicación final, asegurando el correcto funcionamiento como un todo y que la misma realiza las funciones requeridas.

Según el nivel establecido, se seleccionaron dos tipos de pruebas capaces de mostrar la calidad de la aplicación desarrollada. Se pretende además evaluar y verificar las funcionalidades de la misma, así como la seguridad.

Prueba de Seguridad: Verifican que los mecanismos de protección incorporados en el sistema lo protegerán de accesos impropios.

Prueba funcional: Tiene como objetivo asegurar el cumplimiento apropiado de los requisitos funcionales, entrada de datos, procesamiento y obtención de resultados. La principal intención de estas pruebas es medir la correspondencia entre la arquitectura de información propuesta y las funciones que realmente fueron implementadas (50).

Para realizar este tipo de pruebas es necesario tener definidos los requerimientos a verificar con los casos de prueba afiliados a cada uno de ellos. Estos serán los encargados de verificar la aplicación implementada, se utilizará el método de caja negra.

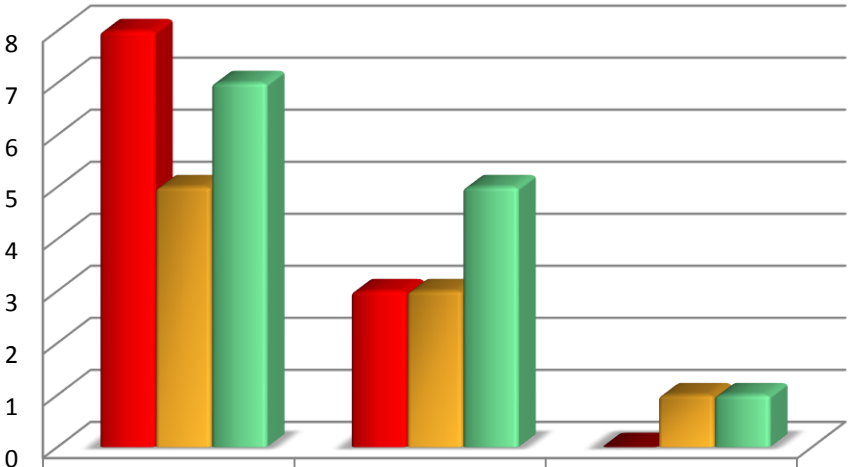
3.3.1 Diseño de casos de prueba.

El método de prueba de caja negra se aplica a la interfaz de la aplicación. Pretende demostrar que las funcionalidades del sistema son operativas, las entradas se aceptan correctamente y que se producen los resultados esperados. Dentro del método de caja negra se utiliza la técnica “Partición Equivalente” siendo considerada como una de las más efectivas en la evaluación de los valores válidos, inválidos, o aquellos valores que no son necesarios para el correcto funcionamiento de la aplicación. La Partición Equivalente

divide el dominio de entrada de un programa en clases de datos a partir de las cuales se derivan casos de prueba (50).

3.3.2 Resultados de las pruebas

Se realizaron tres iteraciones de pruebas, en las que no solo se detectaron varias no conformidades, también surgieron ideas y validaciones necesarias a partir de las recomendaciones de los probadores. En el transcurso de las iteraciones la cantidad de errores tuvo una tendencia a disminuir hasta el punto de en la 3ra iteración notificarse solo una no conformidad no significativa y una recomendación. En la Figura 6 se grafica un resumen de los resultados de las pruebas.



	1ra Iteración	2da Iteración	3ra Iteración
Significativas	8	3	0
No significativas	5	3	1
Recomendaciones	7	5	1

Figura 6: Resumen de las no conformidades arrojadas por las pruebas.

Entre las no conformidades más significativas resaltan las citadas a continuación:

- **Excesivo tiempo para comenzar a mostrar respuestas que ya han sido calculadas**, con motivo de esta no conformidad surge la recomendación de implementar un sistema de entrega en tiempo real que vaya mostrando los dictámenes a medida que estos son calculados.

- **Datos repetidos e inconsistentes con la búsqueda**, como parte de esta no conformidad surge la idea de agregar varios filtros que posibiliten acotar el rango de búsqueda y añadir una optimización extra al tiempo total de ejecución.

Las no conformidades fueron corregidas y originaron nuevos ciclos de pruebas, la mayoría de las recomendaciones fueron aceptadas. En el caso de las dirigidas a factores subjetivos como el rendimiento, se optimizó el tiempo de repuesta, utilizando técnicas de extracción mínima de información en la base de datos y almacenando varios valores anteriormente calculados en cada petición.

3.4 Análisis de los tiempos de ejecución en la detección de plagio distribuida

Para comprobar los tiempos de ejecución alcanzados al distribuir la detección de plagio se realizó un experimento que consistió en aumentar sucesivamente, mil cada vez, la cantidad de detecciones; también se varió la cantidad de nodos utilizados para el procesamiento, aumentando la cantidad progresivamente en una unidad hasta llegar a 16. Las pruebas fueron realizadas en un clúster Beowulf[16] con procesadores “Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz”, memoria RAM “DIMM DDR3 4Gb 1333 MHz”, red “RTL8111/8168/8411 PCI Express Gigabit Ethernet” y sistema operativo “Ubuntu 14.04”.

En la Figura 7 se puede observar como el tiempo de ejecución para 10000 detecciones se logró disminuir de 41597 con una computadora a 3598 milisegundos utilizando 16 computadoras, siendo este un 8.65% del tiempo empleado inicialmente. No obstante, se pudiera pensar que quizás no haya 16 computadoras disponibles para la detección de plagio, pero se logró disminuir el tiempo de detección a 10586 milisegundos utilizando apenas 4 computadoras, siendo este un 25.45% del tiempo empleado inicialmente; además el esquema propuesto permite desplegar detectores en computadoras que no estén dedicadas expresamente a este objetivo pero que su carga de trabajo permita realizar estos procesamientos a la par que son usadas con otros objetivos.

[16] Un **clúster Beowulf** es un sistema de cómputo paralelo basado en clústeres de ordenadores personales conectados a través de redes informáticas.

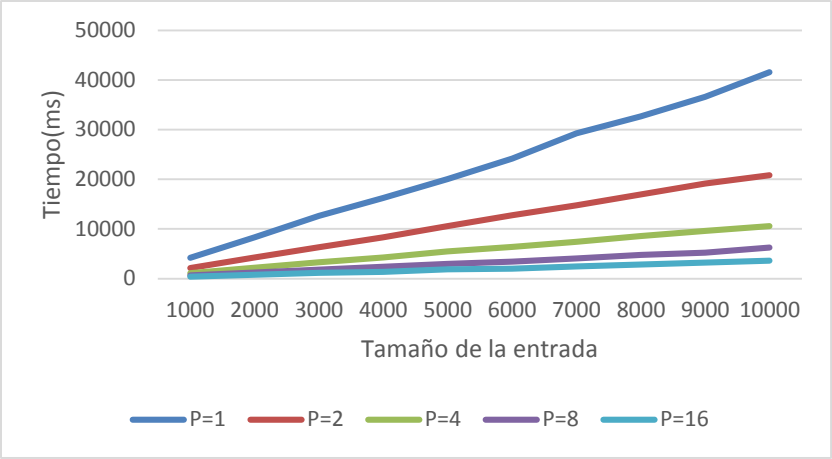


Figura 7: Tiempo de ejecución de la detección de plagio distribuida en 1, 2, 4, 8 y 16 procesadores.

Para valorar el desempeño logrado se calculó y graficó en las Figuras Figura 8 y Figura 9 el *Speedup* algorítmico, definido como la "medida de la mejora de rendimiento de una aplicación al aumentar la cantidad de procesadores comparado con el rendimiento al utilizar un solo procesador" (51). Siendo p la cantidad de procesadores, n el tamaño de la entrada y T el tiempo de ejecución, el *Speedup* S se calcula:

$$S_n = \frac{T_1}{T_n}$$

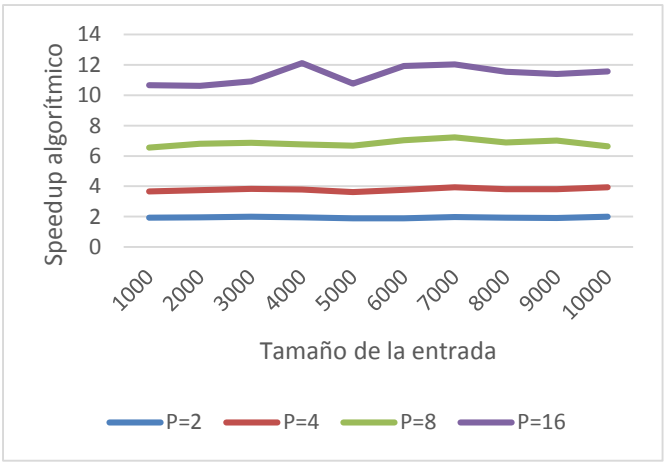


Figura 8: Speedup en función de n.

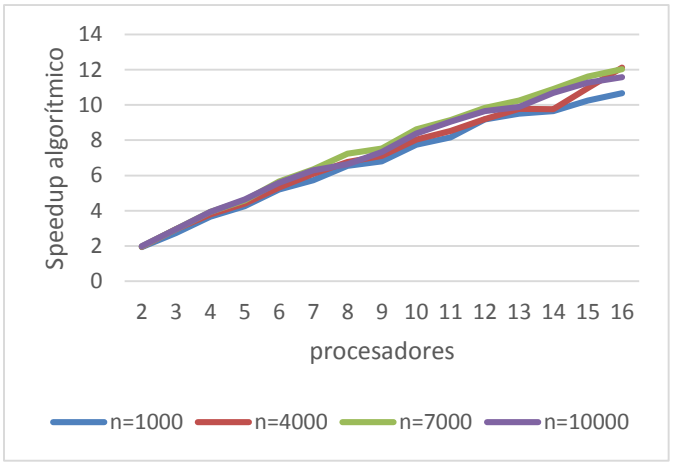


Figura 9: Speedup en función de p.

La eficiencia computacional es un valor normalizado del *Speedup* entre cero y uno. Entiéndase eficiencia como la capacidad del producto software para proporcionar una ejecución o desempeño apropiado, en relación con la cantidad de recursos utilizados, bajo condiciones establecidas, entre los recursos se pueden incluir otros productos del software, la configuración del software y el hardware del sistema y los materiales (52). Valores cercanos a uno indican buena eficiencia. Se calcula mediante la ecuación:

$$E_n = \frac{S_n}{n}$$

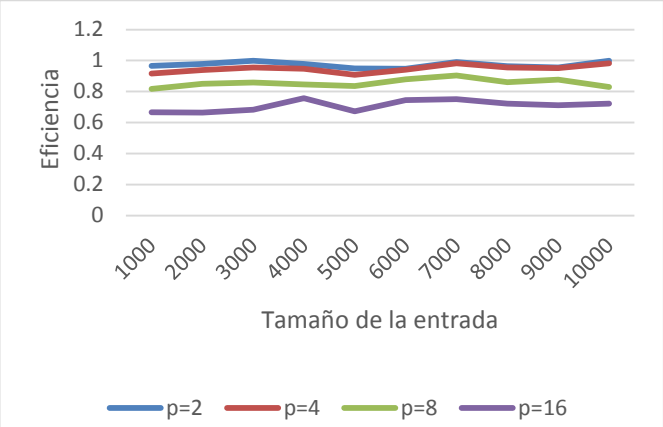


Figura 10: Eficiencia en función de n.

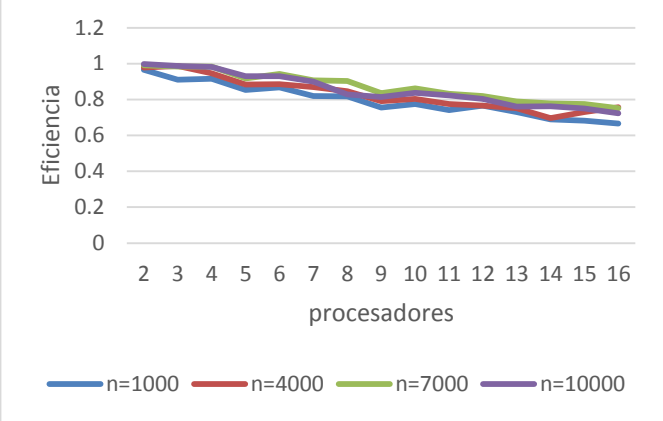


Figura 11: Eficiencia en función de p.

Una eficiencia computacional sobre 0.6 es positiva y en todas las pruebas realizadas los valores fueron superiores; incluyendo eficiencias sobre 0.8 hasta ocho procesadores como se puede observar en las Figuras Figura 10 y Figura 11. Como aspecto negativo se puede apreciar una tendencia a disminuir la eficiencia con la cantidad de procesadores. Analíticamente esto significa que con cada aumento de la cantidad de procesadores se obtendrán mejores tiempos de ejecución, pero en factores cada vez más bajos hasta llegar a un punto en que sea contraproducente dedicar más recursos a la detección de plagio; pues la repercusión en los tiempos de ejecución sería despreciable o negativa. Calculando una línea de tendencia para 10000 detecciones y evaluando en la recta se pudo calcular que la eficiencia se mantendrá positiva hasta alrededor de 50 procesadores.

3.4.1 Múltiples detectores en memoria compartida

Además de aprovechar todas las computadoras disponibles también se desea aprovechar todos los núcleos de procesamiento. Ese objetivo se logrará desplegando varios detectores en una misma máquina, con un máximo de P detectores en una computadora de P procesadores. Para valorar la repercusión de este proceder se experimentó 10 veces para cada cantidad de procesadores, la prueba consistió en realizar 1000 detecciones de plagio y fueron realizadas en una PC dedicada con procesador “Intel(R) Core(TM) i5-M530 CPU @ 2.53GHz”, Memoria RAM “DIMM DDR3 8GiB 1333 MHz” y sistema operativo “Windows 8.1 x64”.

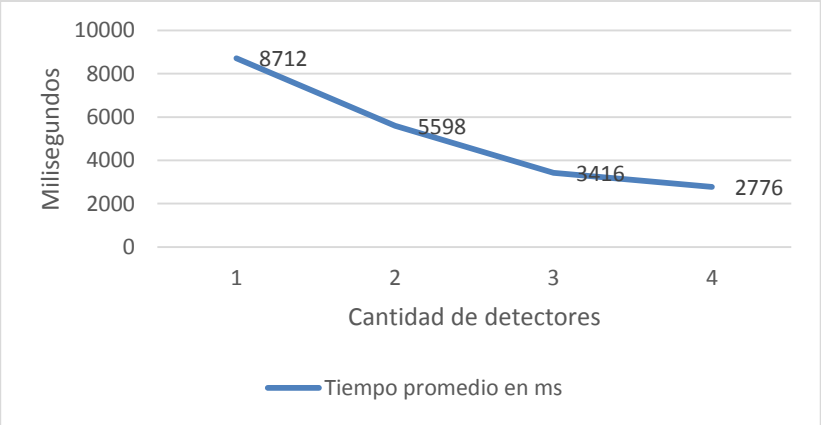


Figura 12: Tiempo de ejecución logrado con la distribución en memoria compartida de la detección de plagio.

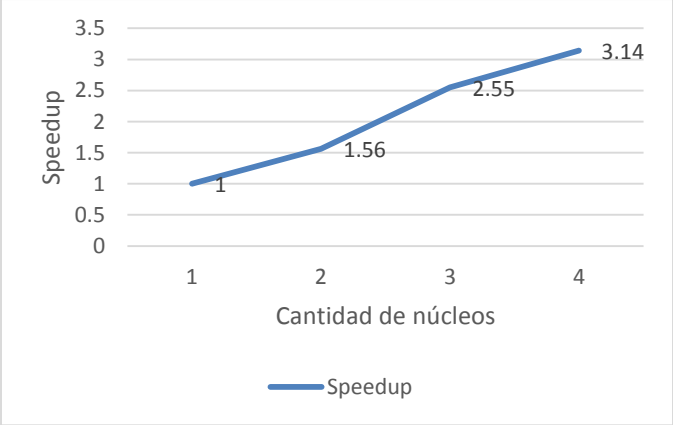


Figura 13: Speedup en función de la cantidad de núcleos.

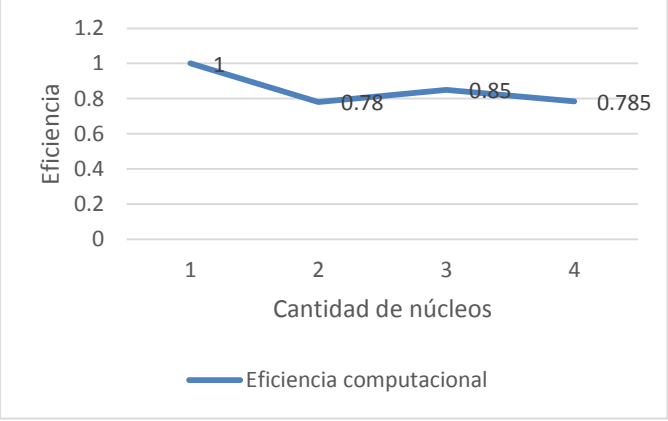


Figura 14: Eficiencia en función de la cantidad de núcleos.

Se puede observar en la Figura 12 como el tiempo de ejecución fue reducido de 8712 milisegundos utilizando un solo detector a 2776 milisegundos utilizando 4 detectores, esto representa un 31.86% del tiempo de ejecución inicial. En la Figura 13 se puede ver reflejada la mejora en los tiempos de ejecución al aumentar la cantidad de procesadores comparado con el tiempo de ejecución al utilizar un solo procesador. En la Figura 14 se puede apreciar el aprovechamiento de los recursos computacionales que se utilizaron para realizar esta prueba.

3.4.2 Sistema de caché distribuida

Redis es una base de datos NoSQL que ha sido ampliamente probada, con un rendimiento respetable utilizado principalmente para el cacheado de contenido. Su uso representa un valor agregado y se traduce en un mejor rendimiento que será aprovechado en el Juez en Línea Caribeño para realizar análisis a una mayor cantidad de datos.

Para comprobar los tiempos de ejecución alcanzados al distribuir la detección de plagio y emplear varias réplicas de Redis para acceder a los códigos fuentes, se llevó a cabo un experimento, que consistió en realizar 10000 detecciones de plagio aumentando sucesivamente la cantidad de computadoras utilizadas para ello. Las pruebas fueron realizadas en un clúster Beowulf con procesadores “Intel(R) Core(TM)2 Duo CPU E4500 @ 2.20GHz”, Memoria RAM “DIMM DDR2 667 MHz” y sistema operativo “Nova 4.0”.

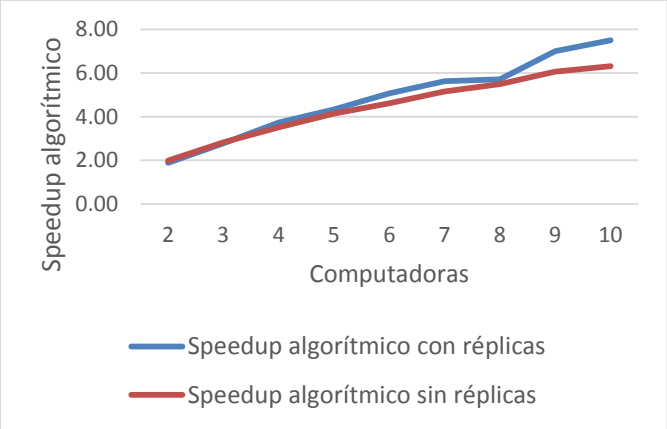


Figura 15: Speedup algorítmico en función de p. Comparación entre instancia única y réplicas de Redis.

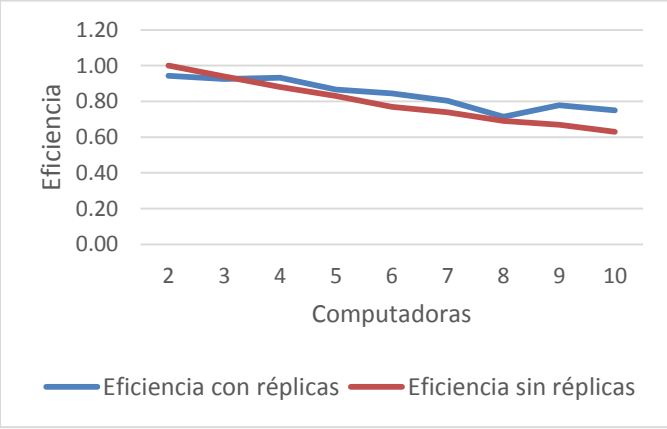


Figura 16: Eficiencia en función de p. Comparación entre instancia única y réplicas de Redis.

En las Figuras Figura 15 y Figura 16 se puede observar la comparación de acuerdo a la aceleración y eficiencia alcanzada en ambas pruebas. La eficiencia general de la detección distribuida utilizando 10 computadoras mejoró un 11.93% con respecto al uso de una sola instancia de Redis.

Para comprobar la repercusión de la utilización de Redis en la solución desarrollada se realizó una comparación realizando 10 pruebas creciendo en número de detecciones.

Las pruebas fueron realizadas en una PC dedicada con procesador “Intel(R) Core(TM) i5-M530 CPU @ 2.53GHz”, Memoria RAM “DIMM DDR3 8GiB 1333 MHz” y sistema operativo “Windows 8.1 x64”.

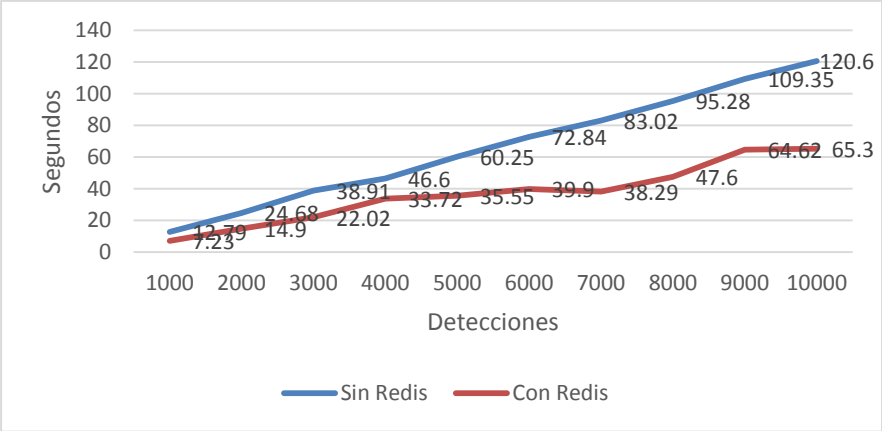


Figura 17: Comparación entre la detección de plagio con y sin Redis

Según los resultados obtenidos en la comparación que se puede observar en la Figura 17, Redis disminuye los tiempos de ejecución en la detección de plagio en un 43% como promedio.

Nótese que el resultado antes expuesto no es una comparación directa entre Redis y PostgreSQL, sino entre la detección de plagio usando Redis y PostgreSQL. En caso de una comparación directa entre ambas bases de datos la diferencia se hace mayor. Probando la obtención de 10000 códigos fuentes usando PostgreSQL versión 9.1 la consulta correspondiente se ejecuta como promedio en 10.720 segundos; no obstante la consulta separada de 10000 códigos fuentes se tardó 111.430 segundos.

Por otra parte, Redis cuenta con la herramienta *redis-benchmark* para la evaluación del rendimiento, esta permite ejecutar determinada cantidad de veces una consulta específica. Se realizó una prueba obteniendo 10000 veces un código fuente; la ejecución de *redis-benchmark* se tardó 0.22 segundos.

La prueba anterior demuestra que Redis, al menos en este contexto, tiene tiempos de ejecución más de 500 veces mejores que PostgreSQL en la obtención de los códigos fuentes necesarios para la detección de plagio.

3.5 Aplicación de la técnica de Iadov

La validación de la solución desarrollada asegura que el producto final cumpla con los requerimientos establecidos para el software, acredita además la calidad y disminución en los tiempos de ejecución del sistema. En la actualidad existen disímiles métodos para determinar la validez de un producto, los cuales presentan diferentes enfoques. Una vez realizadas las pruebas a las interfaces de la aplicación, las entradas y salidas de datos, así como el tiempo de las respuestas, se procede a la validación del grado de aceptación de los clientes. Para ello se aplica la técnica de Iadov para medir el nivel de satisfacción del personal que utilizará el detector implementado.

La técnica de Iadov fue creada originalmente por Kuzmina en 1970, utilizada por López Rodríguez y González Maura en el 2002, para el estudio de la satisfacción por la profesión en carreras pedagógicas. Luego, González Maura y López Rodríguez en ese mismo año realizaron una transformación a esta técnica y la plantean como alternativa para el diagnóstico de la motivación profesional en profesores de Educación Física (53).

Constituye actualmente un camino indirecto para el conocimiento de la satisfacción, se utilizan criterios que establecen relaciones entre tres preguntas que se intercalan dentro del cuestionario y cuya existencia es desconocida por los involucrados. Estas cuestiones se relacionan a través del “Cuadro lógico de V. A. Iadov” (Tabla 6) desarrollado por los citados autores.

Para la recogida de los datos que permitirán conocer la satisfacción, alrededor de la creación y tiempos de ejecución del sistema para la detección de plagio, se confeccionó un cuestionario conformado por diez preguntas, de ellas siete cerradas[17] y tres abiertas[18] que posteriormente van a ser analizadas mediante la técnica V. A. Iadov. El cuestionario se aplica a los administradores del COJ, el Comité de Entrenadores del Movimiento de Programación Competitiva Tomás López Jiménez y varios programadores. Las preguntas están basadas principalmente en la necesidad de la creación de un sistema para la detección de plagio y la

[17] **Pregunta cerrada:** El encuestado está forzado a elegir entre las opciones preestablecidas que se le presentan.
[18] **Pregunta abierta:** El encuestado tiene la libertad de dar su opinión sin tener restricciones.

necesidad de mejorar los tiempos de ejecución en las detecciones de plagio realizadas por los algoritmos implementados.

Tabla 6: Cuadro lógico de V. A. Iadov

5. ¿Qué usted cree del detector de plagio en el Juez en Línea Caribeño?	1. ¿Cree usted que el plagio es un comportamiento correcto?								
	No			No sé			Si		
	3. ¿Cree usted que la dificultad en la detección de plagio hace necesaria la creación de un detector automático que facilite el proceso?								
	Si	No sé	No	Si	No sé	No	Si	No sé	No
Me gusta mucho	1	2	6	2	2	6	6	6	6
No me gusta tanto	2	2	3	2	3	3	6	3	6
Me da lo mismo	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	4	4
No me gusta nada	6	6	6	6	4	4	6	4	5
No sé qué decir	2	3	6	3	3	3	6	3	4

El resultado de la interrelación de las tres preguntas cerradas conlleva a un número que indica la posición de cada uno de los encuestados en la escala de satisfacción siguiente:

1. Clara satisfacción
2. Más satisfecho que insatisfecho
3. No definida
4. Más insatisfecho que satisfecho
5. Clara insatisfacción
6. Contradictoria

La aplicación de la técnica de Iadov y de las preguntas adicionales mostradas en el cuestionario, constituyen un instrumento útil para el estudio y evaluación de la satisfacción-insatisfacción de los encuestados.

Para obtener el índice de satisfacción grupal (ISG) se establece una escala numérica entre +1 y -1 con los diferentes niveles de satisfacción como se muestra en la (Tabla 7) (54).

Tabla 7: Escala numérica de satisfacción

Valor	Interpretación
+1	Máximo de satisfacción
0. 5	Más satisfecho que insatisfecho
0	No definido y contradictorio
-0. 5	Más insatisfecho que satisfecho
-1	Máxima insatisfacción

El índice de satisfacción grupal se calcula por la siguiente fórmula:

$$\mathrm{ISG} = \frac{\mathrm{A}(+1) + \mathrm{B}(0.5) + \mathrm{C}(0) + \mathrm{D}(-0.5) + \mathrm{E}(-1)}{\mathrm{N}}$$

En esta fórmula A, B, C, D, E, representan el número de sujetos con índice individual (1, 2, 3, 4, 5 ó 6) y N representa el número total de sujetos del grupo.

El ISG arroja valores entre +1 y -1. Los valores que se encuentran comprendidos entre - 1 y – 0.5 indican insatisfacción; los comprendidos entre –0.49 y +0.49 evidencian indefinición o contradicción y los que caen entre 0.5 y 1 indican que existe satisfacción.

Estos valores se pueden representar en un eje de coordenada como se aprecia en la Figura 18.

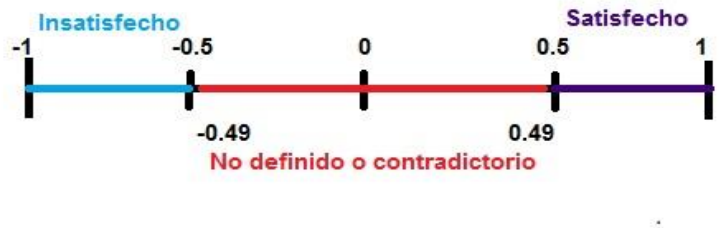


Figura 18: Eje numérico de satisfacción

El procedimiento de cálculo del Índice de Satisfacción Grupal (ISG) de la encuesta realizada a 15 personas queda reflejado a continuación:

$$\mathrm{ISG} = \frac{\mathrm{A}(+1) + \mathrm{B}(0.5) + \mathrm{C}(0) + \mathrm{D}(-0.5) + \mathrm{E}(-1)}{\mathrm{N}}$$

$$\mathrm{ISG} = \frac{10(+1) + 4(0.5) + 0(0) + 1(-0.5) + 0(-1)}{15}$$

$$\mathrm{ISG} = 0.76$$

El resultado arrojado demuestra clara satisfacción con una ISG de 0.76 dentro del rango [0.5, 1].

3.6 Análisis de los resultados

El objetivo final de la validación es contrastar la hipótesis que plantea que: una solución distribuida disminuirá el tiempo de ejecución al detectar plagio en jueces en línea de programación.

La investigación está orientada a la aplicación de una solución distribuida como variable independiente de la hipótesis, evaluada de acuerdo a la cantidad de computadoras empleadas para el procesamiento. El tiempo de ejecución al detectar plagio en jueces en línea de programación, es la variable dependiente que será valorada de acuerdo a la cantidad de detecciones que se logren realizar en un tiempo determinado.

El resultado esperado es que a medida que se aumente la cantidad de computadoras utilizadas disminuirá el tiempo necesario para detectar plagio. Todos los experimentos realizados apuntaron a una disminución superior al 70% con el uso de 4 computadoras para la detección de plagio; a partir de 16 computadoras la disminución alcanza valores superiores al 90%. Los experimentos orientados a la valoración del impacto del sistema de caché distribuida demostraron una disminución adicional en los tiempos de ejecución.

La implementación de la propuesta en el Juez en Línea Caribeño unido a la aplicación de la técnica de Iadov, permitió determinar un nivel de clara satisfacción en las personas encuestadas. La realización de pruebas al sistema permitió identificar y corregir errores funcionales.

El análisis de los resultados derivados de los experimentos realizados, permite concluir que la hipótesis planteada en la investigación fue confirmada. Se cumple el objetivo trazado de desarrollar una solución distribuida para disminuir el tiempo de ejecución al detectar plagio en jueces en línea de programación.

3.7 Conclusiones parciales

En este capítulo se documentó el proceso de validación, se realizaron experimentos que demostraron que se logró disminuir los tiempos de ejecución en la detección de plagio en jueces en línea. Se aplicó también la técnica Iadov para el cálculo de la satisfacción. Se concluye que:

- La implementación de la solución en el Juez en Línea Caribeño cumple con la calidad y los requisitos deseados.
- El procesamiento distribuido utilizando el protocolo AMQP con su implementación RabbitMQ disminuye los tiempos de ejecución al detectar plagio en jueces en línea de programación.
- El uso del sistema de caché distribuida Redis disminuye los tiempos de ejecución en la detección de plagio y disminuye el uso de los recursos computacionales del Juez en Línea, en particular el uso de la base de datos relacional.
- El personal implicado en la detección de plagio en el juez en línea caribeño se encuentra satisfecho con la solución desarrollada.

Conclusiones

Luego de concluir el proceso de desarrollo del detector de plagio documentado en la presente investigación se arribó a las siguientes conclusiones:

- Se realizó una caracterización sobre el plagio en código fuente, que permitió identificar las herramientas y tecnologías necesarias para disminuir el tiempo empleado en su detección.
- Se realizó una caracterización de la detección de plagio en jueces en línea de programación y las herramientas detectoras, que demostró la necesidad de disminuir los tiempos de ejecución.
- Se desarrolló una solución distribuida apoyándose en un sistema de caché, también distribuida, que permitió disminuir los tiempos de ejecución en la detección de plagio.
- Se implementó y probó la solución en el Juez en Línea Caribeño, empleando el *middleware* de mensajería RabbitMQ y el sistema de caché distribuida Redis. Se realizaron un grupo de experimentos para mediar la disminución lograda en los tiempos de ejecución; que junto a la valoración del nivel de aceptación de la propuesta, permitió confirmar la hipótesis planteada.

Recomendaciones

- Implementar detectores que complementen los resultados arrojados por el código fuente y la clasificación del problema, que pueden tener en cuenta métricas de software, comportamiento en tiempo de ejecución, estilo de código de los usuarios, entre otros factores.
- Aplicar el modelo propuesto de distribución de procesamiento y cacheado de contenido en otros procesos llevados a cabo en los jueces en línea, por ejemplo: el motor de calificación, gestión de notificaciones, sistema de trofeos, actualización del ranking.
- Aplicar técnicas de Inteligencia Artificial al consenso de los criterios de los jueces sobre el dictamen y el ajuste dinámico de los pesos de los detectores.
- Desarrollar un mecanismo de entrega en tiempo real de los dictámenes con una tecnología de comunicación servidor-cliente o bidireccional como puede ser WebSocket.

Referencias bibliográficas

1. *Fundamentos de la programacion de computadoras.* **Gutierrez, Andrea.** s.l. : Institución Educativa Simón Bolívar, 2012.

2. *Consejos para tener éxito Programando.* **Perera, Perera, Raydelto HernándezHernández.** Santo Domingo : Raydelto, 2009.

3. **Castilla, Leon.** El Juez Online de la UVA alcanza los diez millones de accesos procedentes de todo el mundo. *Europa PRESS.* 20 Abr, 15 de 2 de 2012, innova-00439. Sacado de http://www.europapress.es/castilla-y-leon/innova-00439/noticia-innova-juez-online-uva-alcanza-diez-millones-accesos-procedentes-todo-mundo-20120420142158.html.

4. *El Jurado en Línea, una nueva forma de ejercitación y evaluación en las asignaturas de programación.* **Junco Vázquez, Tomás Orlando.** Ciudad de la Habana, Cuba : Informática 2009, 2009. ISBN-978-959-286-010-0.

5. **Iniciativa Xtreme Programming.** Xtreme Judge. [En línea] 2006. [Citado el: 26 de 5 de 2012.] http://xtremejudge.uci.cu.

6. **Jimenez, Reymis Monier.** Juez en Línea Cátedra de Programación Avanzada (CPAV). [En línea] 2007. [Citado el: 26 de 5 de 2012.] http://cpav.uci.cu.

7. **COJ development team(CDEVT).** Caribbean Online Judge. [En línea] UCI, 5 de 6 de 2010. [Citado el: 16 de 10 de 2014.] http://coj.uci.cu.

8. **Real Academia Española.** Diccionario de la Lengua Española. *http://buscon.rae.es/draeI/.* [En línea] 2001. [Citado el: 24 de 10 de 2014.] http://buscon.rae.es/draeI/.

9. **ProgramaMe.** ProgramaMe. *ProgramaMe.* [En línea] 21 de 10 de 2014. [Citado el: 21 de 10 de 2014.] http://www.programa-me.com.

10. *Módulo para la detección de plagio en el Juez en Línea Caribeño.* **González Vallejo, Leandro, y otros.** Habana : XV Congreso Internacional de Informática en la Educación, 2013. ISBN: 978-959-7213-02-4.

11. *Sistemas Distribuidos.* **Coulouris, George y Dollimore, Jean.** 3, Rosario, Argentina : Pearson Educacion, 2001. ISBN 8478290494.

12. **Universidad de Oxford.** *http://dictionary.oed.com/.* [En línea] 11 de 2010. [Citado el: 15 de 10 de 2014.] http://dictionary.oed.com/.

13. **Castro, Sonia Jannett Girón.** *Libro anotaciones sobre plagio.* [ed.] Universidad Sergio Arboleda. Primera edición. 2008.

14. **Cedeño, Luis Alberto Barrón.** *Detección automática de plagio en texto.* Universidad Politécnica de Valencia. Valencia : Universidad Politénica de Valencia, 2008. Tesis desarrollada dentro del Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital.

15. *Ph.D. in "Cut-and-Paste"?* **Pew Research Center,.** Washington, D. C. : IWETEL, 2014.

16. *Detección automática de plagio: de la copia exacta a la paráfrasis.* **Barrón Cedeño, Alberto, Vila, Marta y Rosso, Paolo.** p. 76-96, Valencia : Garayzábal Heinze et, 2011.

17. *El plagio y su impacto a nivel académico y profesional.* **Soto Rodríguez, Armando.** 1, San José, Costa Rica : e-Ciencias de la Información, 2012, Vol. 2. ISSN-1659-4142.

18. *JPlag: Finding plagiarisms among a set of programs.* **Prechelt, Lutz, Malpohl, Guido y Philippsen, Michael.** Karlsruhe, Germany : Fakultat fur Informatik, Universitat Karlsruhe, 28 de marzo de 2000, Technical Report 2000-1, University of Karlsruhe, Vols. D-76128.

19. *Curbing epidemic cheating through systemic change.* **Alschuler, A S y Blimling, G S.** 43, s.l. : College Teaching, College Teaching, Vol. 4, págs. 123–126.

20. *El ciberplagi acadèmic: esbrinant-ne les causes per tal d'enllestir les solucions.* **Comas, Rubén y Sureda, Jaume.** 10, s.l. : Digithum, 2008, Digihtum, Vols. p. 1-6.

21. **Universidad de Alcalá.** *Tutorial AlfaBuah.* Biblioteca, Universidad de Alcalá. Alcalá : Biblioteca, 2014. Tutorial.

22. **Enrique Vallés, Balaguer y Rosso, Paolo.** *Empresa 2.0: Detección de plagio y análisis de opiniones.* Valencia : Master's thesis, Universidad Politécnica de Valencia, 2011.

23. *El plagio como ilícito penal.* **Balbuena, Pedro Virgilio.** s.l. : Ventana legal, 2009, Ventana Legal.

24. **Softonic.** Softonic. [En línea] Softonic, 25 de 5 de 2012. [Citado el: 20 de 10 de 2014.] http://onsoftware.softonic.com/detectar-plagio-texto-imagenes-audio.

25. **Wise, Michael J.** *YAP3: Improved detection of similarities in computer program and other texts.* Sydney, Australia : Department of Computer Science, University of Sydney, 1996.

26. *String Similarity via Greedy String Tiling and Running Karp−Rabin Matching.* **Wise, Michael J.** Sydney, Australia : Online Preprint, 1993, Vol. 119.

27. **Kazim, Norulazmi Bin.** *Plagiarims Detection System for Java Programming Assigments by Using Greedy String Tilling Algorithm.* Malaysia : College of Arts and Sciences, 2008. pág. 15, Tesis para obtar por el grado MSc. (ICT).

28. *An algorithmic approach to the detection and prevention of plagiarism.* **Ottenstein, Karl J.** s.l. : ACM SIGSCE, 1976, págs. 30–41.

29. **Wise, Michael J.** Wise's Web Words. [En línea] [Citado el: 24 de 5 de 2013.] http://www.pam1.bcs.uwa.edu.au/~michaelw.

30. *Aplicación de la teoría de grafos y el Algoritmo de Dijkstra para determinar las distancias y las rutas más cortas en una ciudad.* **Restrepo C, Jorge Hernán y Sánches C, John Jairo.** [ed.] Redalyc. 26, s.l. : Scientia Et Technica, 2004, Scientia Et Technica. Disponible en http://redalyc.uaemex.mx/src/inicio/ArtPdfRed.jsp?iCve=84911640021. ISSN 0122-1701.

31. **Sphere Research Labs.** SPOJ. [En línea] Sphere Research Labs, 2012. [Citado el: 27 de 11 de 2014.] http://www.spoj.pl/info.

32. **UCI.** COJ. [En línea] 5 de 6 de 2010. [Citado el: 16 de 10 de 2014.] http://coj.uci.cu/contest/cstatus.xhtml?page=1&cid=1000.

33. **Videla, Alvaro y William, Jason J.W.** *RabbitMQ in Action, Distributed messaging for everyone.* New York : Manning Publications Co, 2012. ISBN 9781935182979.

34. *Advanced Message Queuing Protocol.* **Vinoski, Steve.** 6, IEEE internet computing : IEEE Internet Computing, 2006, Vol. 10, págs. 87-89.

35. **OASIS.** AMQP Advanced Message Queue Protocol. [En línea] OASIS, 2014. [Citado el: 28 de 1 de 2014.] http://www.amqp.org/about/what.

36. **Google.Inc.** Google Trends. [En línea] [Citado el: 20 de 11 de 2014.] http://www.google.de/trends/explore?hl=en-US#q=activemq,+hornetq,+rabbitmq,+qpid&cmpt=q.

37. *Messaging Systems, How to make the right choice?* **Mihailescu, Adina .** Saint-Philippe : Muriel's Tech, 2013.

38. **RabbitMQ.** Rabbit Technologies announces acquisition by SpringSource. [En línea] RabbitMQ, 13 de 04 de 2010. [Citado el: 19 de 05 de 2014.] http://www.rabbitmq.com/news.html.

39. **Fitzpatrick, Brad.** Memcached. [En línea] Dormando, 2009. [Citado el: 24 de 11 de 2014.] http://memcached.org.

40. **Seguin, Karl.** *The Little Redis Book.* Ottawa, Canada : Free Book, 2012.

41. **Citrusbyte.** How fast is Redis? *Redis.IO.* [En línea] Citrusbyte. [Citado el: 13 de 11 de 2014.] http://redis.io/topics/benchmarks.

42. *Proffesional NoSQL.* **Tiwari.** Indianapolis, Indiana : John Wiley & Sons, Inc. 978-0-470-94224-6.

43. **Parr, Terence.** *The Definitive ANTLR 4 Reference.* Dallas, Texas-Raleigh, North Carolina : The Pragmatic Programmers, LLC, 2012 . ISBN 13: 978-1-93435-699-9.

44. **Guerra, Rosy y González, Pamela.** ¿Cómo calcular el promedio ponderado? [En línea] WikiHow, 20 de 11 de 2014. [Citado el: 20 de 11 de 2014.] http://es.wikihow.com/calcular-el-promedio-ponderado.

45. *Correlación y concordancia entre el examen nacional de medicina y el promedio ponderado universitario: análisis de la experiencia peruana en el periodo 2007-2009.* **Huamaní, Charles, Gutiérrez, César y Mezones, Holguín.** 1, Lima, Perú : Revista Peruana de Medicina Experimental y Salud Publica, 2011, Vol. 28, págs. 62-71.

46. *Importancia de la caza para alimentación humana en el curso inferior del río Ucayali, Perú.* **Pierret, Paul V y Dourojeanni, Marc J.** Lima, Perú : Revista Forestal del Perú, 1967, Vol. 1, págs. 10-21.

47. *La valuación de empresas en México Aplicación del modelo de Valor Económico Agregado.* **García Saavedra, María Luisa .** 2004, México : Revista Contaduría y Administración, 2001, Vol. 214, pág. 1.

48. *Moss, A System for Detecting Software Plagiarism.* **Aiken, Alex.** California, USA : Theory Stanford, 2014.

49. *Técnicas para la representación del conocimiento causal: un estudio de caso en Informática Médica.* **MSc. Maikel Leyva-Vázquez, MSc. Karina Pérez-Teruel, Dra. C. Ailyn Febles-Estrada, Dr. C. Jorge Gulín-González.** 1, La Habana : Revista Cubana de información en ciencias de la salud, 2013, Vol. 24.

50. **COJ, development team.** Caribbean Online Judge. [En línea] 5 de 6 de 2010. [Citado el: 24 de 11 de 2014.] http://coj.uci.cu/24h/statistics.xhtml.

51. **Pressman, Roger S.** *Ingeniería de Software, Un enforque práctico.* Séptima. New York : McGraw-Hill, 2010. ISBN 978-0-07-337 597 -7.

52. **Nesmachnow, Sergio y Ares, Gerardo.** *Computación del alta performance.* Grupo de Procesamiento Paralelo Aplicado : CECAL, 2010.

53. **ISO.** *Software product evaluation-Quality characteristics and guidelines for their use. 9126* 2005. ISO.

54. *Las clases de Educación Física y el deporte extraescolar entre el alumnado almeriense de rimaria. Una aplicación práctica mediante la técnica de Iadov.* **Gómez López, Manuel, y otros.** 11, España : Lecturas: Educación Física y Deportes, 2006, Revista Digital - Buenos Aires, Vol. 98.

55. *La técnica de Iadov: Una aplicación para el estudio de la satisfacción de los alumnos por las clases de educación física.* **Rodríguez, Alejandro López y González Maura, Viviana.** Universidad de la Habana, Cuba : s.n., 4 de 2002, Revista Digital - Buenos Aires.

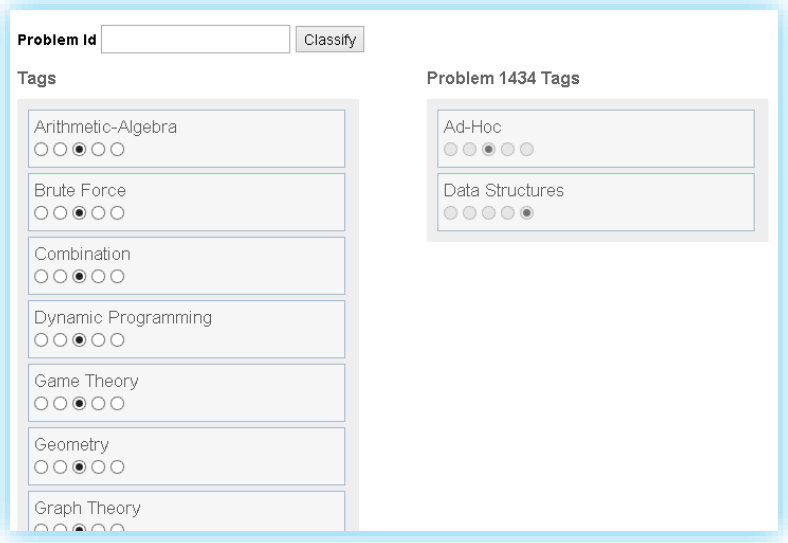
56. **Ruz, Víctor Valenzuela.** *Manual de análisis y diseño de algorítmos.* Copiapó, Chile : Instituto Nacional de Capacitación (INACAP), 2003.

57. *Operating Systems: Three Easy Pieces.* **Arpaci-Dusseau, Remzi H. y Arpaci-Dusseau, Andrea C.** s.l. : Arpaci-Dusseau Books, 2013, Vol. Scheduling Introduction.

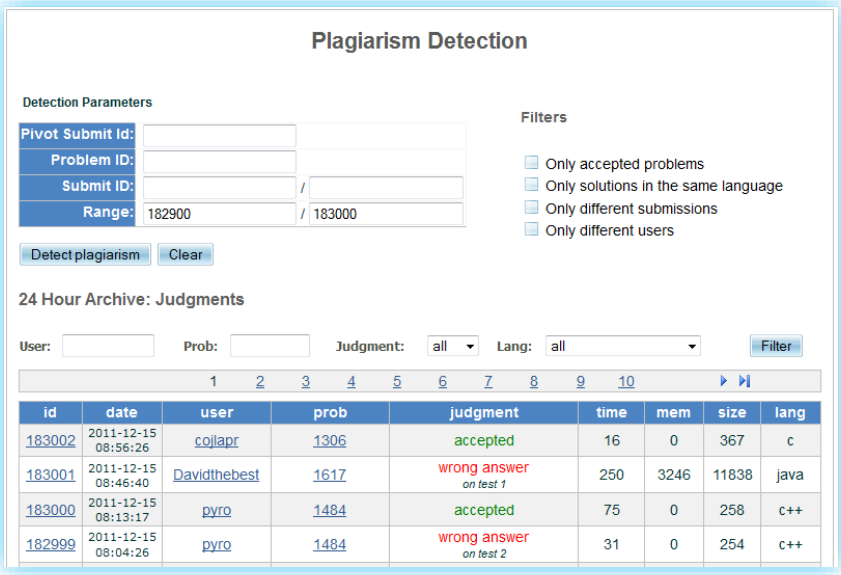
Anexos

```
DECLARE
  curClassification RECORD;
BEGIN
UPDATE classifications SET estimated_code_length = -1;
FOR curClassification IN EXECUTE
    'select * from classifications'
LOOP
    UPDATE classifications SET estimated_code_length = COALESCE((
    SELECT AVG(T.lengthcode -
        (SELECT COALESCE(SUM(estimated_code_length),0) from classifications JOIN
        (SELECT id_classification FROM problem classification WHERE pid = T.pid AND
        id_classification <> curClassification.id_classification ) R ON R.id_classification = classifications.id_classification)
    )
FROM (SELECT LENGTH(code) AS lengthcode,pid FROM submition JOIN source ON sid = submit_id
    WHERE pid IN
        (
            SELECT DISTINCT pid FROM problem classification
            WHERE
                pid IN (SELECT pid FROM problem classification WHERE id_classification = curClassification.id_classification)
            AND
                pid NOT IN
                (SELECT pid FROM problem classification WHERE id_classification <> curClassification.id_classification
                AND id_classification in ( Select id_classification from classifications where estimated_code_length = -1 ) )
        )
    ) T
     ),0) where classifications.id_classification = curClassification.id_classification;
END LOOP;
END;
```

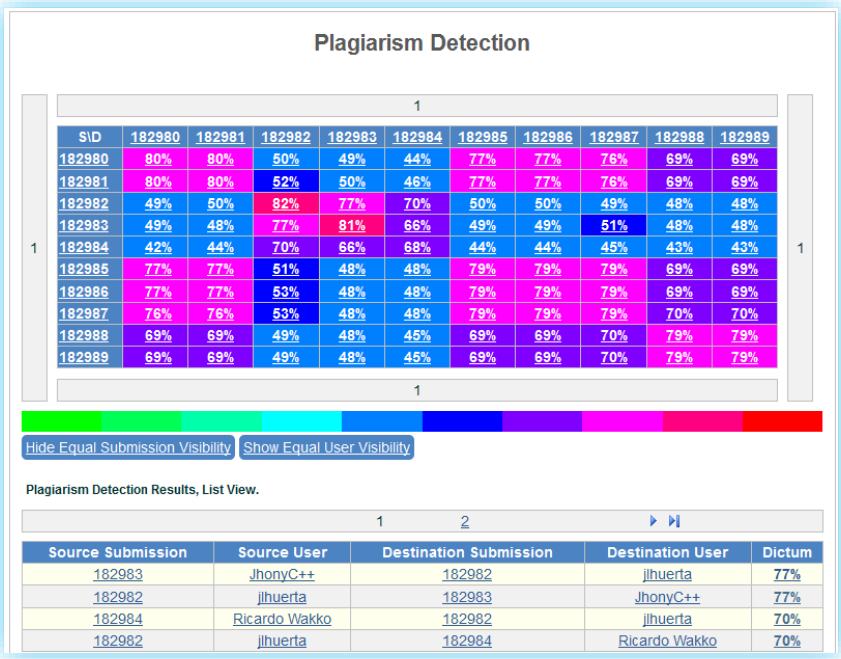
Anexo 2: Procedimiento almacenado update_classification_estimated_code_length que calcula la longitud de código estimada para cada clasificación.



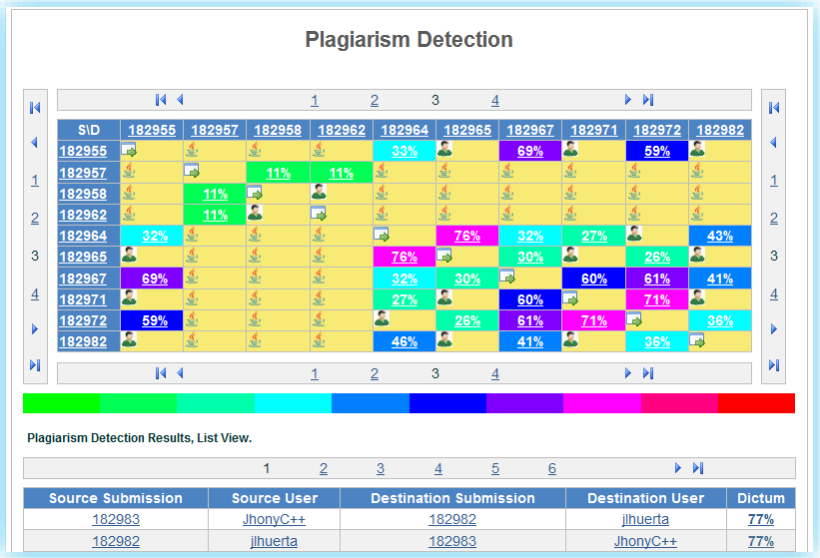
Anexo 1: Vista de la interfaz clasificación de problemas en el COJ.



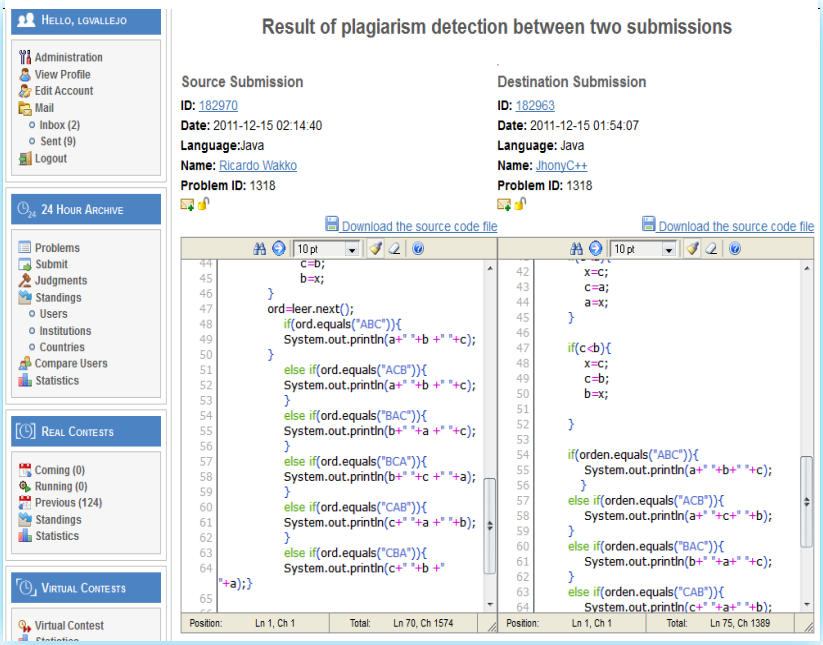
Anexo 3: Vista de la interfaz seleccionar parámetros para la detección.



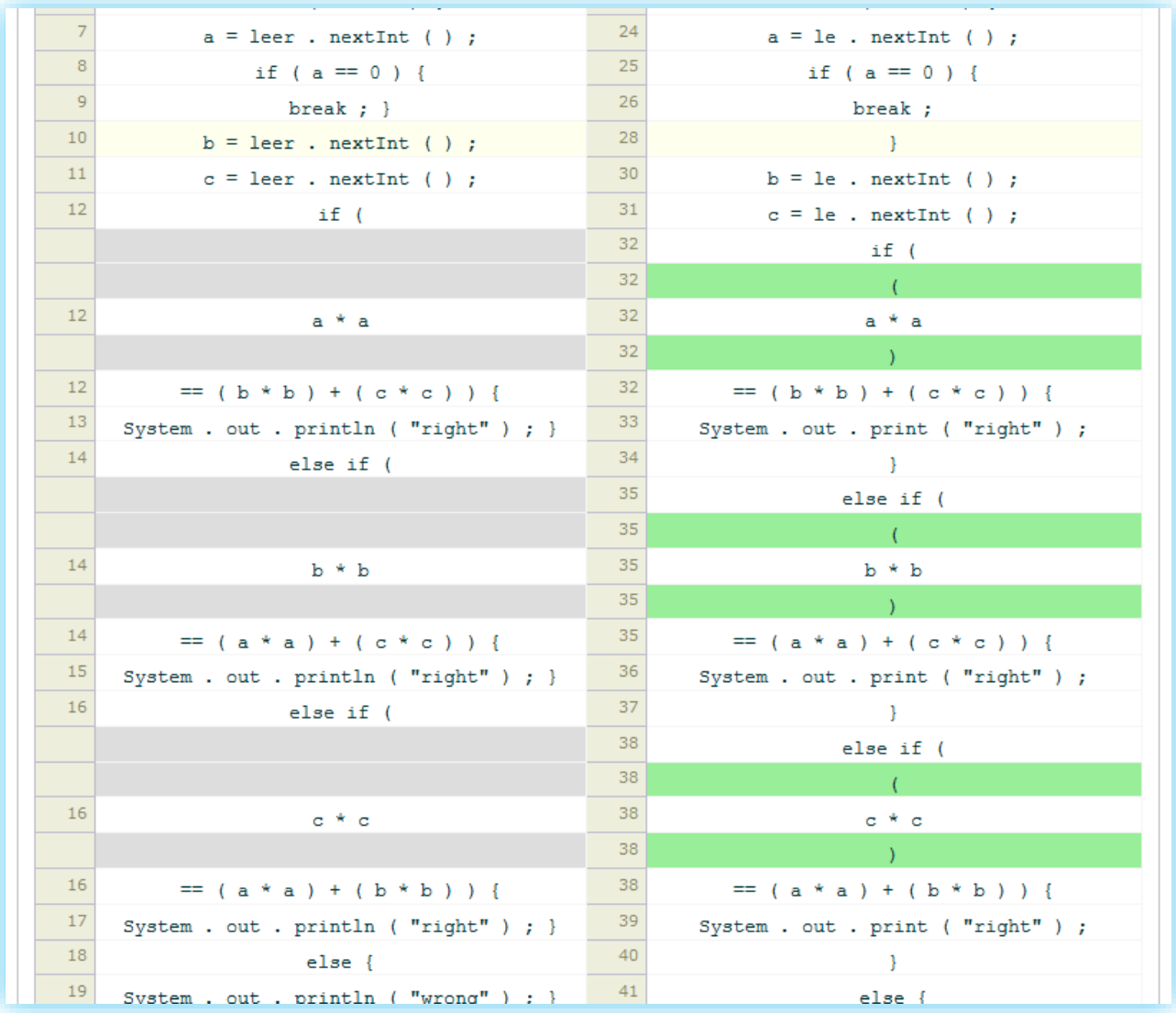
Anexo 4: Vista de la interfaz resultados de la inspección.



Anexo 5: Vista de la interfaz resultados de la inspección con filtros.

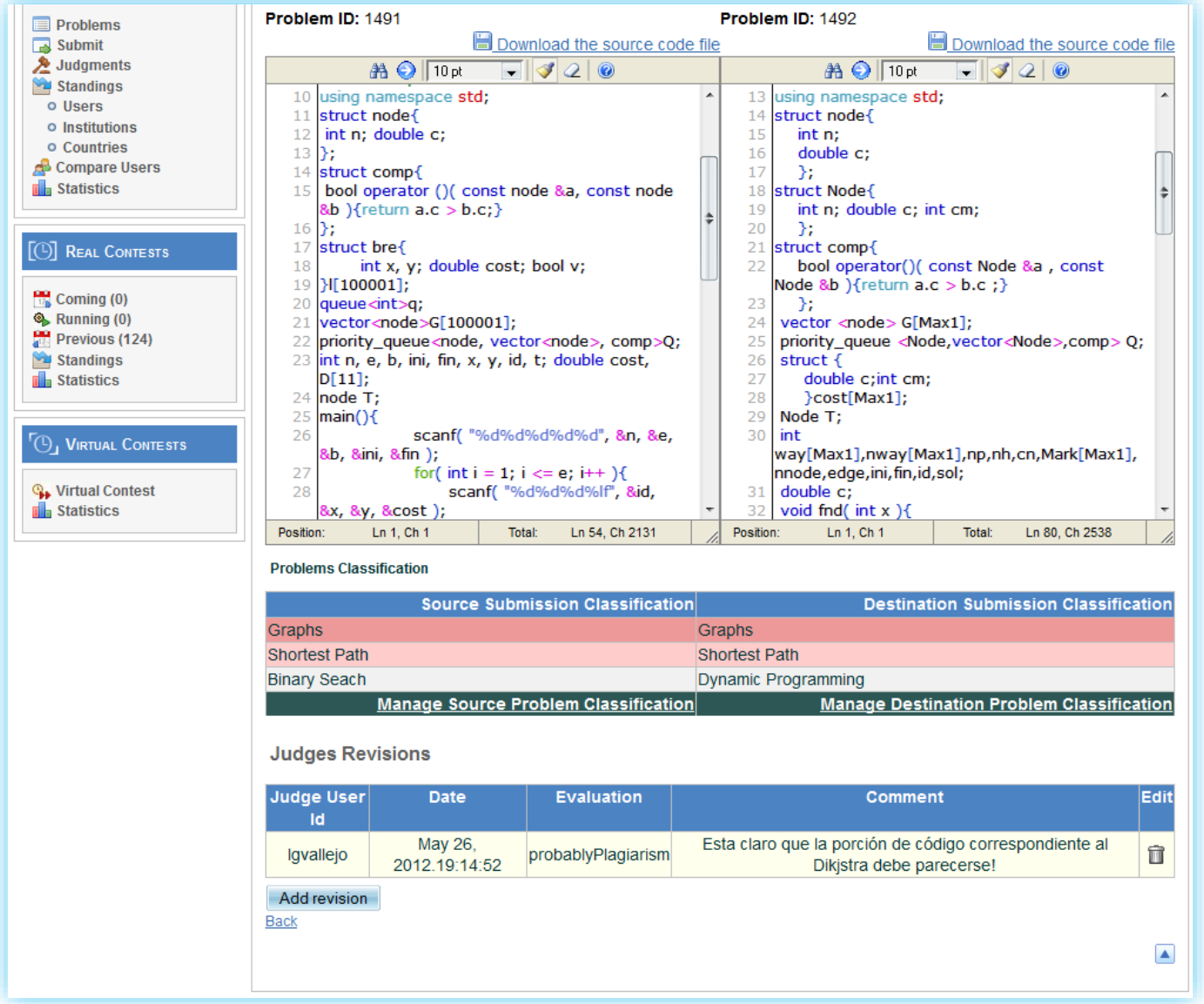


Anexo 6: Vista de la interfaz resultado de la detección de plagio entre dos soluciones.



7	a = leer . nextInt ( ) ;	24	a = le . nextInt ( ) ;
8	if (a == 0) {	25	if (a == 0) {
9	break ; }	26	break ;
10	b = leer . nextInt ( ) ;	28	}
11	c = leer . nextInt ( ) ;	30	b = le . nextInt ( ) ;
12	if (	31	c = le . nextInt ( ) ;
		32	if (
		32	(
12	a * a	32	a * a
		32	)
12	== (b * b) + (c * c)) {	32	== (b * b) + (c * c)) {
13	System . out . println ("right") ; }	33	System . out . print ("right") ;
14	else if (	34	}
		35	else if (
		35	(
14	b * b	35	b * b
		35	)
14	== (a * a) + (c * c)) {	35	== (a * a) + (c * c)) {
15	System . out . println ("right") ; }	36	System . out . print ("right") ;
16	else if (	37	}
		38	else if (
		38	(
16	c * c	38	c * c
		38	)
16	== (a * a) + (b * b)) {	38	== (a * a) + (b * b)) {
17	System . out . println ("right") ; }	39	System . out . print ("right") ;
18	else {	40	}
19	System . out . println ("wrong") ; }	41	else {

Anexo 7: Vista de la interfaz resultado del detector Greedy String Tiling.



Anexo 8: Vista de la interfaz resultado del detector basado en las características del problema.