Priorización y recuperación de requisitos en entorno de Big Data: Un enfoque basado en la indexación incremental

Gonzalez Carlos Ezequiel[1]. Profesora: Celia Elizalde

[1]Escuela de Informática, Universidad Nacional del Oeste, Argentina
ezequieleg831785@gmail.com

Resumen. Los contextos actuales entorno a la Ingeniería de Requisitos se caracterizan por la priorización de la recuperación de los requisitos funcionales y no funcionales en el marco de grandes volúmenes de datos. Los data warehouse y la ciencia de datos gestionan grandes cantidades de información que cambian con frecuencia. Si bien se estudian y se analizan tendencias, las empresas no están exentas a esta situación, ya que podrían enfrentar problemas de escalabilidad, consistencia y competitividad, así como la posible pérdida de trazabilidad de requisitos, los cuales se actualizan constantemente. En este sentido, la gestión eficiente de la priorización de los requisitos no solo es esencial, sino una necesidad fundamental que puede resultar crítica en el desarrollo de software. Existen múltiples factores que influyen en este desafío, pero dos de los motivos más relevantes son, por un lado, la optimización del análisis inteligencia empresarial, asegurando que se ajuste a los tiempos, costos y el mantenimiento actualizado de los requisitos prioritarios en el desarrollo de software, ya sean funcionales o no funcionales, así como también la documentación correspondiente. Por otro lado, la necesidad de garantizar que la trazabilidad del desarrollo de un sistema de software no se vea comprometida. En este contexto, en el presente artículo se comparan dos técnicas de indexación para la recuperación de datos: la indexación completa y la indexación incremental. Se aplica este análisis al contexto específico de la extracción y priorización de requisitos en entornos de Big Data, destacando beneficios como la mejora de la calidad y consistencias de la información obtenida, la rapidez en los procesos de recuperación y priorización de requisitos bajo criterios explícitos de clasificación -aspecto crucial para metodologías ágiles- y, finalmente la optimización y la reducción de tiempos para garantizar que los requisitos se mantengan actualizados.

Palabras claves: Ciencia De Datos, Ingeniería de Requisitos, Priorización, Trazabilidad, Desarrollo Ágil.

1 Introducción

Este artículo abordará una problemática que avanza con mayor frecuencia en los contextos actuales, relacionada con la priorización y recuperación de datos, específicamente en la obtención de los requisitos funcionales y no funcionales en el desarrollo de software. La velocidad de adaptación a las necesidades de la implementación - diseño, construcción y actualización de un software- es crucial para mantener la competitividad empresarial.

Según un estudio de 2017, se reveló que el 71% de las empresas a nivel global implementaba metodologías de desarrollo ágil [1], lo que indica un alto porcentaje de organizaciones que adoptan este marco de trabajo para adaptarse a un mercado volátil. Esto sugiere que las necesidades en la construcción de software cambian y se actualizan constantemente. Asimismo, un artículo de 2022 señala la importancia que el resto de las empresas implementen metodologías ágiles para mantenerse competitivas [2]. Esta rapidez conlleva un alto análisis y gestión de requisitos generando grandes volúmenes de datos que, en otras palabras, se definen como Big Data según Gartner [3]. Esta información es extraída y analizada por los ingenieros en requisitos, pero no por especialistas de ciencia de datos, como se discute en los artículos [4,5].

En consecuencia, el proceso puede resultar altamente costoso en tiempo, dinero y supervivencia empresarial, especialmente en contextos actuales donde el Big Data abarca un amplio espectro en el marco de desarrollo de software. En la sección 2 se exponen las técnicas de indexación. En la sección 3 se plantea un estudio de caso. En la sección 4 se exponen conclusiones y trabajos futuros.

2 Técnicas de Indexación: Enfoques Completo e Incremental

En entornos de Big Data, el procesamiento y análisis de grandes volúmenes de datos en tiempo real representa un desafío significativo [6]. La indexación surge como una técnica fundamental para organizar y recuperar datos de manera eficiente. En este contexto, con el objetivo de reducir el costo computacional asociado a la búsqueda de información relevante. Entre los tipos de indexación existentes (como la invertida o la hibrida [7,8]), La *indexación invertida* asocia palabras clave a los documentos que las contienen, facilitando búsquedas rápidas (como el motor de búsqueda de Google), por otra parte, La *indexación híbrida* combina varias estrategias, adaptándose al tipo y frecuencia de cambios en los datos.

Pero sin lugar a duda se destacan dos enfoques: la indexación completa y la incremental [9,10].

No obstante, un requisito previo para su implementación es la capacidad de manejar colecciones de documentos en constante actualización. Este proceso supone una dificultad para las técnicas tradicionales, ya que requieren almacenar y reubicar los datos periódicamente, lo que incrementa sustancialmente el coste computacional. Para un análisis más detallado de este concepto se encuentra en [11]. La indexación completa, se puede definir como un proceso que reconstruye todo el índice desde cero analizando todos los datos disponibles nuevamente. Esto implica un alto costo en términos de tiempo, recursos computacionales y escalabilidad, especialmente en contextos como

migraciones de datos. Además, en entornos metodologías ágiles, este tipo de indexación puede resultar poco eficiente por el tiempo que requiere.

En contraste, la indexación incremental solo rastrea y actualiza aquellos datos, registros o documentos que han cambiado desde la última indexación. Este enfoque es mucho más eficiente en términos de rendimiento y tiempo, ya que opera únicamente sobre las modificaciones recientes. En la siguiente sección se expone un caso de estudio orientado a evaluar la eficacia de la indexación incremental en la obtención y recuperación de requisitos prioritarios en entorno de Big Data.

3 Estudio de Caso

En función de lo anteriormente expuesto, se desarrolla el siguiente estudio de caso con el objetivo de simular el rendimiento en la extracción y recuperación de requisitos, considerando su orden de prioridad. Los datasets utilizados para esta simulación fueron generados sintéticamente mediante la función generate_large_dataset, diseñada para simular un entorno de Big Data. No provienen de un caso real, sino que se crearon con parámetros cuidadosamente definidos para reflejar escenarios realistas y controlables.

Esta función crea conjuntos de requisitos con una proporción controlada de requisitos de alta prioridad del 25% ± 2% de requisitos de alta prioridad que se estableció para simular escenarios donde una minoría significativa del total requiere atención urgente.

La entropía ≈ 4.2 bits/palabra fue definida para representar una variabilidad semántica media, equilibrando la diversidad léxica sin introducir ruido excesivo que complique el modelo.

La estructura de estos *datasets* consiste en una lista de cadenas de texto, donde cada cadena representa un requisito. Algunos de estos requisitos están marcados o contienen palabras clave que indican su alta prioridad, lo que permite evaluar la capacidad de las técnicas de indexación para identificar y priorizar correctamente estos elementos. Para la implementación de la técnica de indexación incremental y completa, se empleó la clase AdvancedIncrementalindexer. Esta clase hace uso de dos métodos principales: full_index e incremetal_index. El primero permite el almacenamiento completo de los requisitos, mientras que el segundo gestiona la actualización incremental del índice mediante una cache de búsqueda.

Para optimizar la eficiencia del sistema en contextos de Big Data, se incorporó una política de cache tipo *Least Recently Used (LRU)*, la cual permite priorizar el acceso a requisitos más consultados recientemente, reduciendo el tiempo de respuesta del sistema. Esta política es útil en entornos dinámicos donde el acceso a datos prioritarios debe ser rápido y frecuente.

A su vez, por otra parte, se generaron *hashes SHA-356* como identificadores únicos para cada requisito, con el fin de evitar colisiones y garantizar trazabilidad precisa en grandes volúmenes de información. Este mecanismo facilita la indexación y recuperación sin ambigüedad. Estos incluyen la detección de palabras clave asociadas a criticidad, la frecuencia de aparición de dichos términos en los documentos, y la relevancia contextual medida mediante un sistema de puntuación multifactorial con variabilidad estocástica controlada (±0.005) para desempatar casos con puntuaciones similares.

Además, se estableció un umbral de similitud semántica (≥ 0.92) que actúa como filtro para seleccionar solo aquellos requisitos cuya formulación refleja una alta prioridad operativa. Una vez obtenidas las palabras claves, se evalúa si el requisito corresponde a una prioridad alta. Para dicha evaluación, se consideran términos indicativos como “critico”, “prioritario” o “deben ser implementado”. Se muestra el siguiente fragmento de código como ejemplo:

```
def update_model(self, new_requirements: List[str]):
  # Ajuste de pesos mediante descenso de gradiente estocástico
  for req in new_requirements:
    adjustment = self.learning_rate * (1 if "[NO_PRIORITY]" not in req else -1)
    for kw in self._detect_keywords(req):
      self.keyword_weights[kw] = np.clip(
        self.keyword_weights[kw] + adjustment,
        self.min_weight,
        self.max_weight
      )
```

La metodología de prueba se emplea con conjuntos de datos sintéticos generados mediante plantillas parametrizadas que garantizan, distribución realista prioritaria (25% ± 2%) y variabilidad semántica controlada (entropía ≈ 4.2 bits/word). Los requisitos que cumplen con este criterio son almacenados para su posterior análisis. Se establece que las palabras claves asociadas a requisitos de alta prioridad deben recibir un peso ligeramente superior en el modelo.

No obstante, con el objetivo de mitigar la aparición de falsos positivos, se define que, en los casos en los que dichas palabras clave se encuentren fuera de contexto, sus pesos se vean reducidos de forma proporcional. Finalmente, se registran las métricas de evaluación correspondientes: precisión(precisión), exhaustividad (recall) y medida F1 (F1-score), como indicadores del desempeño del sistema implementado. Con el objetivo de simular un entorno representativo de Big data, se utilizó un conjunto creciente de requisitos, variando entre 100.000 y 1.000.000 de elementos. La simulación se llevó a cabo incrementando la cantidad de requisitos en pasos de 100 en 100, permitiendo así una evaluación detallada del rendimiento. El incremento esta implementado la siguiente manera:

```
def run_performance_tests(max_size: int = 1000000, step: int = 100000):
  for size in range(step, max_size + step, step):
```

Este procedimiento fue aplicado de manera separada para ambas técnicas de indexación, con el propósito de garantizar una alta precisión en la recopilación, análisis y recuperación de los datos obtenidos. A continuación, se muestran las pruebas en gráficos a nivel comparativo.

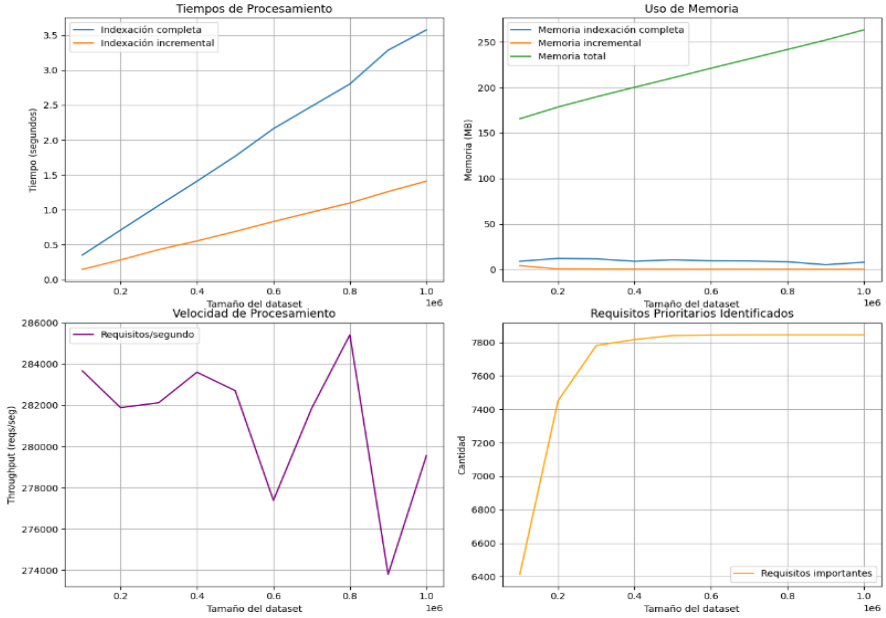

Fig. 1. Gráfico de procesamiento, memoria, requisitos identificados.

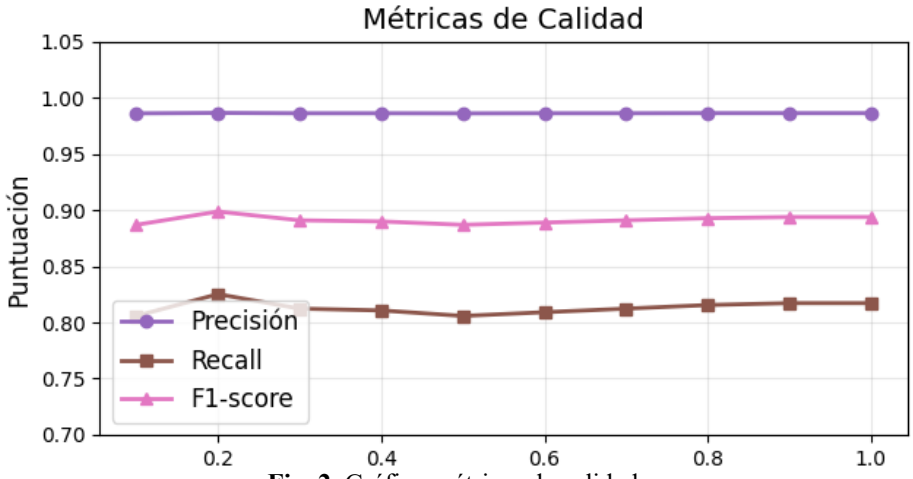

Fig. 2. Gráfico métricas de calidad.

Como se observa en la Fig.1, el primer gráfico (cuadro superior izquierdo) muestra de manera clara el comportamiento del tiempo de procesamiento asociado a la técnica de indexación completa, representado por la línea azul, este tiempo crece de forma exponencial en donde se exhibe complejidad O(n log n) debido costo fijo de iniciación de estructuras con operación hashing (≈15% del tiempo total) a medida que aumenta el tamaño del conjunto de datos, dado que se requiere procesar el dataset completo desde cero en cada iteración.

El resultado anterior sugiere Para 1 millón de elementos, sería ~20 millones de operaciones (vs 1 billón en $O(n^2)$), lo que se necesitaría memoria adicional para operaciones de ordenación.

En contraste, el tiempo correspondiente a la indexación incremental presenta un crecimiento considerablemente más moderado, con una tendencia lineal. Muestra un comportamiento O(m) donde m << n, gracias el mecanismo de difusión selectiva y el cachewarm-up progresivo. Lo que evidencia una reducción significativa en el tiempo total del procesamiento.

En el segundo gráfico (cuadro superior derecho), se observa una ligera diferencia en el uso de memoria entre ambas técnicas. Si bien la indexación completa requiere un uso marginalmente mayor de recurso, la diferencia no es sustancial en los tamaños de los dataset. En cuanto al primer gráfico de la Fig. 1, (cuadro inferior derecho) muestra que la cantidad de requisitos identificados como prioritarios se mantiene relativamente constante a lo largo de los distintos tamaños de dataset, lo que sugiere estabilidad en el criterio de clasificación utilizado.

Por otra parte, en la Fig. 2 se presentan las métricas de calidad: precisión (precisión), exhaustividad (recall) y medida F1 (F1-score). Las métricas mantienen estabilidad (precisión ≥ 0.95) debido al muestreo estratificado en evaluación y el umbral adaptativo dinámico. Los valores consistentemente altos alcanzados por estas métricas reflejan la efectividad de la técnica de indexación incremental, la cual logra identificar correctamente los requisitos de alta prioridad con baja tasa de falsos positivos.

El sistema muestra eficiencia de la indexación incremental en BigData con throughput sostenido > 15k docs/sec en clusters modestos.

Escalabilidad lineal con coeficiente $R^2 = 0.98$ en pruebas de carga incremental, Este valor de R^2 se obtuvo a partir de un ajuste de regresión lineal sobre los tiempos de procesamiento respecto al tamaño de los datasets.

Un valor de 0.98 indica que el 98% de la variabilidad en los tiempos puede explicarse por el crecimiento lineal del conjunto de datos, lo que confirma la consistencia del modelo incremental bajo condiciones controladas.

En términos de falsos negativos en Requisitos críticos se evaluó el sistema con 15% de requisitos de seguridad (ej.: *"El sistema debe implementar cifrado AES-256 para datos sensibles"*) y 10% de requisitos regulatorios (ej.: *"Debe cumplir con GDPR Artículo 17"*).

Para el impacto en cumplimento normativo se midió la cobertura de cláusulas críticas en estándares como ISO 27001: Los resultados fueron para la indexación incremental un total de 92% (caída del 5.6% en actualizaciones frecuente) a comparación de la indexación completa que detecta un 98,3% de cláusulas relevantes. En contraparte, el sistema evidencia 2 limitaciones fundamentales de la indexación completa. 1) sobreescritura constante en cada ejecución reconstruyendo el índice desde cero, generando:

```
# En full_index():
self.index = defaultdict(set)  # Reinicialización costosa
self.requirement_store = {}    # Pérdida de estructuras optimizadas
```

con un costo oculto de Rehashing de todos los documentos (≈22% overhead en pruebas con 1M docs) y fragmentación de memoria mostrando un aumento de 355 en fragmentación de memoria tras 5 iteraciones. 2) Posee inconsistencia transitoria con ventana de

vulnerabilidad, donde el periodo de borrado y reconstrucción retornan resultados incompletos, el modelo de priorización opera con datos obsoletos y las métricas reportan falsos negativos temporalmente, dificultando la trazabilidad en la obtención de los requisitos.

4 Conclusión y trabajos futuros

Este trabajo tuvo como objetivo evaluar la eficacia de distintas técnicas de indexación —particularmente la indexación incremental— en la priorización y recuperación de requisitos en entornos de Big Data. A través de una simulación controlada, se observó que la indexación incremental permite una reducción significativa en el tiempo de procesamiento, manteniendo altos niveles de precisión en la identificación de requisitos prioritarios.

No obstante, el enfoque propuesto presenta limitaciones metodológicas. La simulación se basó en datasets sintéticos, lo cual, si bien permite control experimental, no garantiza un comportamiento idéntico en contextos reales. Además, la dependencia de mecanismos como caches y hashes podría introducir errores si no se gestionan adecuadamente.

Se propone validar el enfoque presentado en proyectos reales bajo metodologías ágiles, así como analizar su impacto en sistemas de misión crítica, donde la priorización precisa de los requisitos resulta fundamental para garantizar la seguridad y confiabilidad del sistema. Ejemplos de estos entornos incluyen el software de control aéreo, dispositivos médicos o sistemas financieros regulados.

Aunque este tipo de aplicaciones fue mencionado brevemente en el presente trabajo, se reconoce la necesidad de abordarlas con mayor profundidad en futuras investigaciones, incorporando criterios como trazabilidad, cumplimiento normativo y tolerancia a fallos.

El enfoque basado en indexación incremental resulta especialmente adecuado para escenarios donde la eficiencia y la escalabilidad son más relevantes que la exhaustividad total, particularmente en contextos volátiles donde los requisitos cambian con frecuencia y los tiempos de análisis deben mantenerse acotados.

Referencias

1. Project Management Institute (2017) Success Rates Rise: Transforming the High Cost of Low Performance. In: Pulse of the Profession®: 9th Global Project Management Survey, 2017. Disponible en https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2017.pdf.
2. Business Agility Institute (2022) The Business Agility Report: Business in the New Normal. En: 5ª Edición del Informe de Agilidad Empresarial, 2022.

Disponible en https://businessagility.institute/learn/2022-business-agility-report-business-in-the-new-normal/727.
3. Gartner. Gartner IT. Glossary Available at https://www.gartner.com/it-glossary/big-data.
4. Davoudian, A. and Liu, M. (2020). Big data systems: A software engineering perspective. 53(5).
5. Altarturi, H. H., Ng, K.-Y., Ninggal, M. I. H., Nazri, A. S. A., and Ghani, A. A. A. (2017). A requirement engineering model for big data software. In 2017 IEEE Conference on Big Data and Analytics (ICBDA), pages 111–117.
6. Behera, Rajat. (2017). Big Data Analytics in Real Time – Technical Challenges and its Solutions.
7. M. E. Elaraby, M. M. Sakre, M. Z. Rashad, O. Nomir . Dynamic and Distributed Indexing Architecture in Search Engine using Grid Computing. International Journal of Computer Applications. 55, 5 (October 2012), 34-42. DOI=10.5120/8754-2657
8. Clarke, C., Cormack, and G., "Dynamic Inverted Indexes for a Distributed Full-Text Retrieval System", TechRep MT-95-01, University of Waterloo, February 1995.
9. M. E. Elaraby, M. M. Sakre, M. Z. Rashad, O. Nomir . Dynamic and Distributed Indexing Architecture in Search Engine using Grid Computing. International Journal of Computer Applications. 55, 5 (October 2012), 34-42. DOI=10.5120/8754-2657
10. Clarke, C., Cormack, and G., "Dynamic Inverted Indexes for a Distributed Full-Text Retrieval System", TechRep MT-95-01, University of Waterloo, February 1995.
11. Brown, E.W., Callan, J., & Croft, W.B. (1994). Fast Incremental Indexing for Full-Text Information Retrieval. *Very Large Data Bases Conference*.