# Indexación Distribuida con Restricción de Recursos

Tomás Delvechio
Departamento de Ciencias Básicas
Universidad Nacional de Luján
tdelvechio@unlu.edu.ar

Gabriel Tolosa
Departamento de Ciencias Básicas
Universidad Nacional de Luján
tolosoft@unlu.edu.ar

***Abstract*—En la actualidad, las organizaciones de todo tipo y tamaño tienen a su disposición grandes volúmenes de información a muy bajo costo. Aún más, aumentan su capacidad de generar datos y, por consiguiente, una necesidad intrínseca de almacenarlos y procesarlos. El paradigma actual para el gestión de datos masivos, conocido como Big Data, ofrece técnicas, algoritmos y plataformas desarrollados específicamente en este contexto. Una de las plataformas más utilizadas es Hadoop y su implementación del paradigma de programación MapReduce. En este trabajo se aborda el problema de la indexación distribuida con restricción de recursos. Se cuenta con un número máximo de nodos de computo con características de equipo de escritorio. Los experimentos variando el tamaño de la entrada muestran que se escala linealmente en las configuraciones estudiadas aunque el speedup resultante es bajo.**

## I. Introducción

En la actualidad, las organizaciones de todo tipo y tamaño tienen a su disposición grandes volúmenes de información a muy bajo costo. Aún más, aumentan su capacidad de generar datos y, por consiguiente, una necesidad intrínseca de almacenarlos y procesarlos. El último tiempo ha mostrado un rápido crecimiento en la disponibilidad de conjuntos de datos (*datasets*) en casi cualquier disciplina, desde ciencia e ingeniería hasta negocios, pasando por salud, política, leyes y economía, entre otros [10]. El tamaño y la escala de los *datasets* pueden ser abrumadores y no solamente son cada vez más complejos y heterogéneos sino que, además, sólo están aumentando.

En muchos casos, el almacenamiento de grandes volúmenes de datos es un primer problema que habilita luego diferentes tareas entre las que se destacan, principalmente, diferentes tipos de búsquedas. Desde hace varios años, el desarrollo de sistemas distribuidos y paralelos ha impulsado múltiples aplicaciones masivas, tanto en datos como en usuarios. Un ejemplo extremo son los motores de búsqueda web (WSE), que procesan miles de millones de páginas[1] en fracciones de segundos para retornar a los usuarios la respuesta a una consulta [4]. Uno de los desafíos consiste en escalar eficientemente con conjuntos de datos en constante crecimiento.

El paradigma actual para el gestión de datos masivos, conocido como Big Data [1], ofrece técnicas, algoritmos y plataformas desarrollados específicamente en este contexto. Una de las plataformas ampliamente adoptada es Hadoop [11], que utiliza el paradigma de programación MapReduce, que facilita el procesamiento de grandes conjuntos de datos distribuyendo la carga de trabajo en múltiples máquinas de un cluster. Incluso, proveedores de servicios de computación en la nube ofrecen de forma transparente servicios Hadoop/MapReduce escalables. La idea original de esta plataforma es poder distribuir la carga de trabajo entre equipos de escritorio con recursos limitados (*commodity hardware*). Sin embargo, diferentes trabajos exploran el uso de Hadoop en clusters o, incluso, en servicios de computación en la nube en la cual la forma mas sencilla de resolver el problema de escalabilidad es simplemente agregar mas nodos de cómputo.

En el caso de indexación distribuida, la tarea consiste en tomar la colección de documentos como entrada y generar las estructuras de datos necesarias que forman el índice invertido [2] y soportan luego la búsqueda eficiente. La indexación es una tarea de uso intensivo de disco ya que se deben recuperar todos los documentos y luego almacenar el índice resultante. Las principales restricciones en este proceso están asociadas al hardware disponible [7]. Resulta claro entonces que cuando el tamaño de la entrada (colección) sobrepasa los recursos disponibles en un único equipo, se requiere procesar de forma distribuida y es aquí donde la plataforma Hadoop/MapReduce facilita la tarea. Además, WSE de gran escala se ejecuta en múltiples máquinas, el resultado final es un índice invertido particionado en varias *shards*, en general, una para cada máquina que participe de la búsqueda.

### A. Motivación y Contribuciones

En este trabajo se aborda el problema de la indexación distribuida con restricción de recursos. Es decir, se cuenta con un número máximo de nodos de computo con características de equipo de escritorio, o *low end hardware* [11], sin la posibilidad de agregar equipos cuando el problema crece. Si bien hay trabajos previos sobre este proceso en el entorno Hadoop/MapReduce [3, 6, 8], en general no cuentan con las restricciones mencionadas.

Si bien la industria ya plantea que los problemas de Big Data exceden los recursos de hardware commodity[2,3], el escenario propuesto se lo considera real ya que, por ejemplo, laboratorios de universidades o pequeñas organizaciones poseen un

[1] De acuerdo a World Wide Web Size (http://www.worldwidewebsize.com/) el tamaño del índice de Google contiene alrededor de 48.000 millones de documentos.

[2] http://www.infoworld.com/article/2610477/big-data/big-data-demands-more-than-commodity-hardware.html

[3] https://www.linkedin.com/pulse/hadoop-big-data-longer-runs-commodity-hardware-david-bennett

número limitado de equipos *low-end* y puede aprovecharlos de todos modos. Además, la solución de utilizar servicios de computación en la nube no siempre es posible por algún tipo de restricción. Por ejemplo, ciertos datos sensibles de algún ente (como el gobierno) no podrían ser procesados en los datacenters de tales proveedores ya que, en general, se encuentran fuera de la jurisdicción de muchos países, con lo cual aparecen involucrados aspectos legales.

Como contribuciones, se presentan los resultados preliminares de un estudio de escalabilidad sobre un problema de indexación real en el cual se generan índices particionados bajo una estructura de datos denominada Block-Max [4] y utilizando compresión de datos, ambas estrategias comúnmente usadas en sistemas de recuperación de información de gran escala. Se propone un algoritmo de indexación y una configuración del cluster acorde a los recursos con los que se cuenta y se estudia la escalabilidad con diferentes tamaños de entrada.

El resto del artículo se encuentra organizado de la siguiente manera: la Sección II presenta conceptos preliminares y los trabajos relacionados. La Sección III introduce el algoritmo utilizado, mientras que la Sección IV presenta los experimentos y los resultados obtenidos. La Sección VI ofrece conclusiones preliminares y trabajos futuros.

## II. Preliminares y Trabajos Relacionados

Para poder responder consultas de forma eficiente, un sistema de Recuperación de Información (RI) construye un índice sobre la colección $C$ de documentos (índice invertido). De forma simple, se construyen dos estructuras de datos: un lexicón y un archivo de *posting lists*. El primero almacena el vocabulario de la colección, es decir, el conjunto de todos los términos únicos que aparecen en documentos y son de interés mantener. El segundo, almacena las ocurrencia de cada término en los documentos que aparece e información adicional utilizada para el ranking (por ejemplo, la frecuencia $f_{i,j}$ de un término $t_i$ en un documento $d_j$).

El enfoque de construcción de índices invertidos en un contexto distribuido es un tema ampliamente tratado en la literatura [9]. También esta problemática es abordada desde la perspectiva de los datos masivos utilizando algoritmos basados en MapReduce [3] y posteriormente estudiado mediante el uso de Hadoop [6] [8].

En la propuesta original de MapReduce una de las implementaciones propuestas es la de un algoritmo para construcción de índices invertidos [3]. Es importante destacar que al usar MapReduce la colección se encuentra previamente particionada entre los nodos debido al sistema de archivos distribuido subyacente. Además, la plataforma se encarga de la asignación de procesos y del procesamiento local.

Trabajos posteriores ofrecen comparaciones de diferentes algoritmos MapReduce para indexación con otros esquemas mas complejos [8]. El algoritmo original de Dean y otros [3] hace un uso intensivo de la red de forma innecesaria. Es posible mejorar el proceso Map agregando procesamiento parcial del documento, como se propone en el algoritmo de Nutch [8]. Lin [6] utiliza características avanzadas de Hadoop para plantear un algoritmo ligeramente diferente. Teniendo en cuenta que MapReduce realiza una operación de ordenamiento en la fase intermedia de forma nativa a partir de las claves, plantea el uso de claves compuestas, donde el motor puede ordenarlas.

En todos los casos, se trata de índices standard donde no siempre se especifica si se utiliza o no compresión. En la actualidad, una de las estructuras de índice competitivas para máquinas de búsqueda es Block-Max [4]. Los autores proponen una estructura auxiliar que permite al algoritmo de búsqueda incorporar técnicas de terminación temprana para reducir el tiempo insumido en la búsqueda.

## III. Indexación con Restricción de Recursos

Aquí se propone una variante de Map (Algoritmo 1) y Reduce (Algoritmo 2) propia que considera el enfoque provisto por McCreadie [8] y suma optimizaciones propuestas en Lin [6] sobre aspectos de escalabilidad de algoritmos de indexación sobre MapReduce.

**Algoritmo 1:** Pseudo-código para las función Map del algoritmo de indexación.

```
Entrada: Clave: Identificador de documento
         Valor: Contenido del Documento
Salida : Clave: Par (Termino, Documento)
         Valor: Frecuencia

para Term ← DocumentContent.nextTerm() hacer
    posting ← Posting(Term, Document);
    postingDocs.agregar(posting);
para posting ← postingDocs.nextPosting() hacer
    emitir(posting.clave, posting.frecuencia);
```

**Algoritmo 2:** Pseudo-código para las función Reduce del algoritmo de indexación.

```
Entrada: Clave: Par (Termino, Documento) parKey
         Valor: Lista de Frecuencias frequencies
Salida : Clave: Término
         Valor: Lista de Postings

postingList ← new PostingList();

term ← parKey.getTerm();
docId ← parKey.getDoc();

si term ≠ termAnt entonces
    emitir(termAnt, postingList);

para freq ← frequencies hacer
    totalFreq ← totalFreq + freq;

posting.addPosting(docId);
termAnt ← term;
```

El algoritmo presentado requiere de memoria suficiente para un solo documento en un nodo durante la fase Map, lo

que facilita su escalabilidad. En el caso del proceso Reduce, la implementación actual almacena la posting list completa para un término en memoria. Sin embargo, debido a la implementación del patrón value-to-key [6], es posible en un trabajo futuro hacer escalable el mismo emitiendo las listas cuando el proceso este en el limite del consumo de memoria, dado que al llegar ordenadas, la construcción de las *posting lists* es la adición de los elementos en la medida que son recibidos.

## IV. Experimentos y Resultados

**Datos:** Se utiliza un subconjunto de una colección real de documentos web provista por la universidad de Stanford (Proyecto WebBase[4]), que ocupa 60 GB de espacio en disco. Con el objetivo de estudiar el comportamiento conforme aumentan los datos de entrada, se generaron dos instancias mas duplicando (120 GB) y triplicando (180 GB) la misma colección (denominadas C1, C2 y C3, respectivamente). Las *posting lists* duplican y triplican su tamaño, lo cual es interesante a los efectos de la prueba ya que la longitud de éstas es una de las cuestiones a tener en cuenta en el algoritmo de indexación.

**Hardware:** Se utiliza un cluster de equipos low-end compuesto por 7 nodos con procesador Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz (4 Cores), 8Gb de memoria principal y disco rígido SATA de 500GB, todos conectados a una red Gigabit Ethernet. Para la configuración de los parámetros de Hadoop se utiliza una herramienta provista por Hortonworks[5] que se basa en la cantidad de núcleos, memoria y cantidad de discos de los nodos. Dado que se trata de discos "comunes", se configura el factor de replicación con el valor por defecto (3 réplicas por bloque) para garantizar la disponibilidad de los datos.

**Configuraciones:** Se generan cuatro configuraciones que resultan de dos tipos de índice invertido diferente (standard y block-max) y dos variantes (comprimido y sin comprimir). Para la versión comprimida se utilizan codecs standard para índices invertidos: PForDelta para los docIDs y Simple16[5] para las frecuencias asociadas. Como la compresión se realiza en la etapa final, cuando ya se cuenta con la*posting list* completa, el proceso impacta en la carga de trabajo de los reducers. En cuanto al cluster, se ejecutan experimentos con 1, 2, 4 y 6 nodos (más un nodo master en todos los casos) variando la cantidad de procesos reducers (también 1, 2, 4 y 6 reducers, cuando corresponde). Además, se realiza una ejecución stand-alone (local) de cada configuración y sus correspondientes variantes, tomadas como procesamiento secuencial.

## V. Métricas

Se utilizan como métricas el tiempo total de procesamiento junto con dos métricas de rendimiento clásicas en la evaluación de sistemas de cómputo paralelo: speedup y eficiencia. El speedup es una medida de la mejora de rendimiento de la aplicación al aumentar la cantidad de procesadores (comparado con el rendimiento al utilizar un solo procesador) y se define como: $S(p) = \frac{T_s}{T_p}$, donde es $T_s$ es el tiempo requerido por el algoritmo secuencial para resolver el problema y $T_p$ el tiempo insumido por el algoritmo paralelo utilizando $p$ procesadores. Por otro lado, la eficiencia cuantifica el speedup obtenido por procesador y se define como $E(p) = \frac{S_p}{p}$ donde $S_p$ es el speedup utilizando $p$ procesadores.

Además, para ser justos con la comparación se resguarda que la salida del algoritmo secuencial sea exactamente la misma que la de los algoritmos paralelos. Para este problema se asume que la máquina de búsqueda que usa los índices generados está formada por $p'$ procesadores. Por simplicidad, se asume $p' = p$, por lo cual la cantidad de procesadores para indexar y luego realizar la búsqueda es la misma. Por ello, asumiendo un cluster de $p$ procesadores en cada configuración, el índice centralizado generado por el algoritmo es partido en $p$ partes y cada una transferida a un nodo. Este costo es adicionado al proceso secuencial.

### A. Resultados

En primer lugar, se presentan los tiempos totales requeridos para la tarea bajo las diferentes configuraciones. Por restricciones de espacio se muestran solo los resultados para el índice Block-Max (el mas complejo) en su versión comprimida.

La Tabla I muestra los tiempos al generar un índice comprimido. La nomenclatura se refiere a la cantidad de nodos de cómputo (N) y de reducers (R) de cada configuración. La primera observación inesperada es que los tiempos son similares al índice sin comprimir. Al generar el índice comprimido ocurren dos cuestiones: por un lado, hay un mayor procesamiento en los reducers ya que deben ejecutar el codec de compresión pero, por otro lado, el tamaño resultante es mucho menor (aproximadamente un 40% respecto del índice original) lo que requiere una menor cantidad de bytes escritos a disco (y transferidos por la red). En principio, estos costos se compensan, generando tiempos similares. No obstante, esta situación requiere una mirada mas profunda para comprender completamente el tradeoff entre tales procesos.

| | C1 | C2 | C3 |
|---|---|---|---|
| **0N-1R** | 26486 | 53051 | 82703 |
| **1N-1R** | 27074 | 56190 | 84285 |
| **2N-1R** | 11230 | 22847 | 34271 |
| **2N-2R** | **11104** | **22751** | **34127** |
| **4N-1R** | 6629 | 13465 | 21214 |
| **4N-2R** | 5488 | 10900 | 22328 |
| **4N-4R** | **5189** | **10233** | **15129** |
| **6N-1R** | 5646 | 11860 | 20009 |
| **6N-2R** | 4086 | 7877 | 12293 |
| **6N-4R** | 3409 | **6914** | **10043** |
| **6N-6R** | **3159** | 7128 | 12359 |

Tabla I: Tiempos totales (en segundos) para las distintas configuraciones al generar un índice comprimido

Otra observación es que para la colección C1 la configuración con el número máximo de nodos y reducers resultó la mas eficiente, mientras que para C2 y C3, 4 reducers ofrece la mejor performance. A priori, la cantidad óptima de reducers

[4]http://diglib.stanford.edu:8091/~testbed/doc2/WebBase/

[5]http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.0/

está en función de la cantidad de discos físicos con que cuenta cada nodo (para permitir escrituras en paralelo) aunque si la replicación se encuentra activada (como en este caso), algunas porciones de los archivos de salida se transmiten a través de la red. Una inspección rápida permite tener una primera explicación relacionada con que la cantidad de información que circula por la red debido principalmente a la replicación, comienza a tener impacto en la performance con un mayor volumen de datos.

La Figura 1 muestra el speedup alcanzado [6]. Para ello, se toma la mejor configuración de cada grupo de acuerdo al número de reducers (cuando la cantidad de nodos $> 1$). Para la cantidad de procesadores $p$ se toma en número total de núcleos disponibles en cada configuración (4 por nodo). Esto es una aproximación pesimista ya que si bien Hadoop utiliza varios, la configuración sugerida reserva un núcleo para el sistema de archivos HDFS y otro para el sistema operativo. Esto se pudo corroborar observando la carga de CPU promedio que se encuentra alrededor del 50%. Mas allá de cuántos núcleos se tomen para el cálculo, lo que muestra la figura es que el speedup se mantiene lineal al crecer el tamaño de la entrada para los casos de prueba.

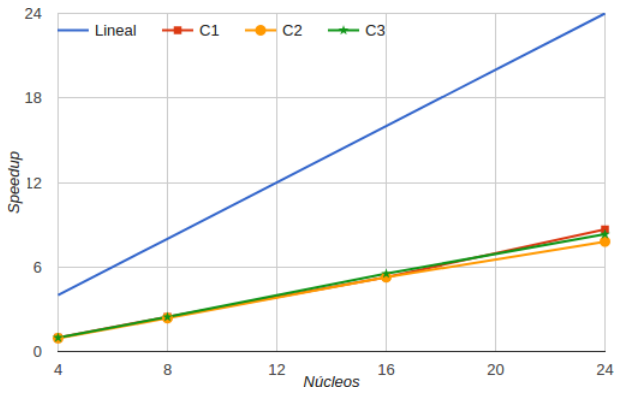


Fig. 1: Speedup de la mejor configuración variando la cantidad de nodos para los diferentes tamaños de la entrada (C1..C3)

Un estudio más detallado podría considerar dos alternativas: utilizar todos los núcleos para procesar y analizar el impacto en los procesos HDFS y del sistema operativo o bien, utilizar 3 núcleos por nodo y evaluar si efectivamente el speedup se incrementa un 25%. En resumen, el análisis actual solo considera 2 núcleos/nodo pero se decide presentar pesimista ya que el núcleo ocioso no es aprovechado de otra manera por el framework. Finalmente, teniendo en cuenta las consideraciones mencionadas, la eficiencia varía entre 23 y 36%, existiendo un incremento interesante del 25% entre las configuraciones de 1 a 2 nodos (23% vs 29%).

## VI. Conclusiones

En este trabajo inicial se aborda el problema de la indexación distribuida para soportar un sistema de búsqueda con restricción de recursos (utilizando hardware low-end). Se generan índices comprimidos y sin comprimir variando la cantidad de nodos de cómputo y el tamaño de la entrada. Los experimentos muestran que el algoritmo utilizado escala linealmente variando el tamaño de la entrada para diferentes configuraciones de nodos y reducers. Sin embargo, el speedup resultante es bajo y, por lo tanto, la eficiencia (con un máximo del 36%).

Los trabajos en curso pretenden, por un lado, mejorar el speedup haciendo un mejor aprovechamiento de los núcleos de los procesadores a partir de una configuración más fina. Por otro lado, se pretende establecer las cotas de rendimiento impuestas por el tráfico de red a partir de establecer parámetros de replicación menores o nulos, en función de la performance de los discos.

[6]Se muestra solo el speedup para la versión comprimida del índice.

## Agradecimientos

Los autores agradecen al Centro de Investigación, Docencia y Extensión en TIC de la UNLu (CIDETIC, http://cidetic.unlu.edu.ar/) por proveer lo recursos computacionales para el proyecto.

## References

[1] ITU-T Study Group 13. Big data - cloud computing based requirements and capabilities. recommendation y.3600, 2015.

[2] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: the concepts and technology behind search*. Addison-Wesley, 2nd edition, 2011.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *OSDI*, 2004.

[4] Shuai Ding and Torsten Suel. Faster top-k document retrieval using block-max indexes. In *Proc. of the 34th SIGIR, international conference on Research and development in information retrieval*. ACM Press, 2011.

[5] Daniel Lemire and Leonid Boytsov. Decoding billions of integers per second through vectorization. *Software - Practice and Experience*, 45(1):1–29, 2015.

[6] Jimmy Lin and Chris Dyer. Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177, 2010.

[7] Christopher Manning, Prabhakar Raghavan, Hinrich Schütze, et al. Introduction to information retrieval. *An Introduction To Information Retrieval*, 151, 2008.

[8] Richard McCreadie, Craig Macdonald, and Iadh Ounis. Comparing distributed indexing: To mapreduce or not? *7th Workshop on Large-Scale Distributed Systems for Information Retrieval*, i(July):41–48, 2009.

[9] Berthier Ribeiro-Neto, Edleno Moura, Marden Neubert, and Nivio Ziviani. Efficient distributed algorithms to build inverted files. In *Proc of the 22nd SIGIR, international conference on Research and development in information retrieval*, pages 105–112. ACM, 1999.

[10] Viktor Mayer Schönberger and Kenneth Cukier. *Big data: la revolución de los datos masivos*. Turner, 2013.

[11] Tom White. *Hadoop: The definitive guide*, volume 54. O'Reilly Media, Inc., 4th edition, 2015.