UNIVERSIDAD CENTRAL DE VENEZUELA

FACULTAD DE CIENCIAS

ESCUELA DE COMPUTACIÓN

UNA APLICACIÓN PARA RECONOCIMIENTO ÓPTICO DE CARACTERES (OCR) BAJO EL ENFOQUE DE COMPUTACIÓN EN LA NUBE

Trabajo Especial de Grado

presentado ante la ilustre

Universidad Central de Venezuela por los

Br. Antonio Enrique Rivero Martínez y

Br. Evander Gene Palacios Cordero

para optar al título de

Licenciado en Computación

Tutora: Claudia León

Caracas, Mayo de 2012

Una Aplicación para Reconocimiento Óptico de Caracteres (OCR) bajo el enfoque de computación en la nube

Autor: Antonio Enrique Rivero.

Autor: Evander Gene Palacios.

Tutora: Claudia León.

Fecha: Mayo de 2012.

RESUMEN

El objetivo del presente Trabajo Especial de Grado (T.E.G.) es explotar las capacidades de la computación en la nube (Cloud Computing en inglés) para mejorar el rendimiento de una aplicación encargada del proceso de reconocimiento óptico de caracteres (OCR, Optical Character Recognition), llevado a cabo sobre documentos del Banco Central de Venezuela, en el marco de un proyecto existente entre esa institución y la Facultad de Ciencias de la Universidad Central de Venezuela.

Para cumplir este objetivo se desarrolló un servicio que cuenta con la plataforma de CloudCrowd, la cual es una gema del lenguaje Ruby, para brindar soporte al ambiente de computación en la nube. Posteriormente se integró el servicio en la nube con la aplicación web de administración del proyecto, respetando el formato de salida existente para minimizar el impacto en los otros procesos de la aplicación original, ya que este T.E.G. está enfocado únicamente en el OCR. La comunicación entre la aplicación web y el servidor en la nube se realiza con el uso de gemas que implementan el formato JavaScript Object Notation (JSON) y el protocolo Secure Copy (SCP).

Palabras Clave: Digitalización, Reconocimiento Óptico de Caracteres, Computación en la Nube, Ruby On Rails, CloudCrowd.

Tabla de contenido

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En los últimos años la digitalización de la información (textos, imágenes y sonidos) ha sido un punto de gran interés para la sociedad, principalmente para fines de conservación de activos, control de procesos y difusión de información en la web. En el caso concreto de los textos, se generan continuamente gran cantidad de información escrita, tipográfica o manuscrita. En este contexto, poder automatizar la introducción de caracteres evitando la entrada por teclado, implica un importante ahorro de trabajo humano y una mejora de la calidad de muchos servicios.

En este sentido, la Facultad de Ciencias de la UCV desarrolló una aplicación para digitalizar y publicar en la web diferentes documentos del Banco Central de Venezuela desde el año 1941 hasta la actualidad. El proceso de digitalización comprende los pasos mostrados en la Figura 1.

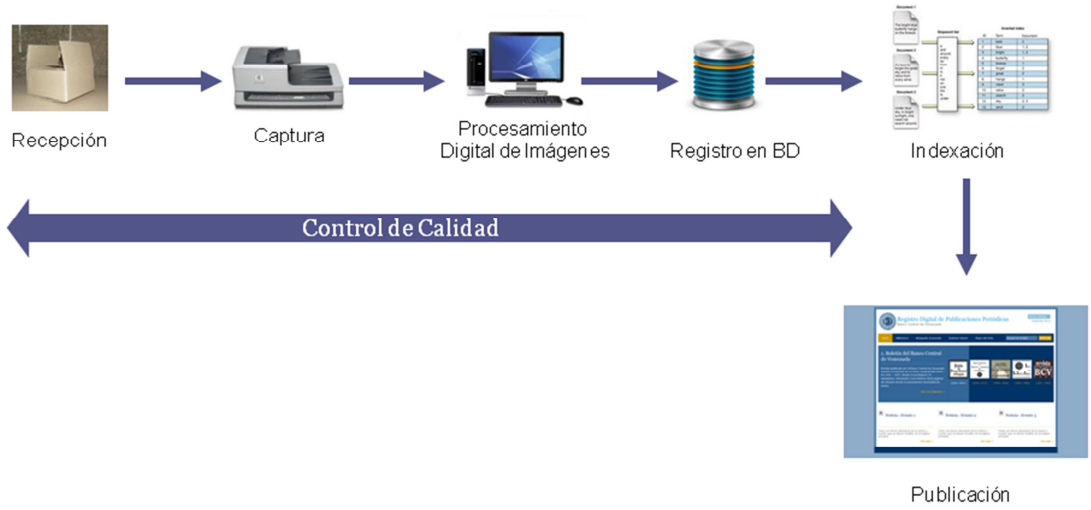

Figura 1. Proceso actual de Digitalización de Documentos del BCV

Específicamente el proceso de indexación comprende las siguientes acciones:

- Elaboración de una taxonomía temática a partir del contenido de los documentos.
- Clasificación de los artículos en base a la taxonomía.
- Uso de software de reconocimiento óptico de caracteres y un clasificador probabilístico, para generar un índice de búsqueda por contenido de los artículos.

Para este Trabajo Especial de Grado se toma como caso de estudio la herramienta existente que permite el reconocimiento óptico de caracteres, para así explorar las potencialidades de la tecnología de computación en la nube para elevar el rendimiento de los procesos OCR.

En este documento se describe el desarrollo de un software como servicio SaaS (Software as a Service), que permite procesar el OCR de documentos publicados por el Banco Central de Venezuela, empleando como base la aplicación web original del proyecto. Esta fue modificada para su integración con el servicio. También se diseñaron pruebas para medir el tiempo de procesamiento de cada revista enviada al servicio, así como pruebas de precisión de resultados en cuanto a la salida obtenida por el servicio en la nube.

Este documento está estructurado en 3 capítulos, además de esta introducción. En el primer capítulo se presentan los conceptos básicos que contextualizan el trabajo, así como las herramientas utilizadas para el desarrollo. En el segundo capítulo se describe en detalle el software desarrollado y los experimentos diseñados para las pruebas. Igualmente se analizan los resultados obtenidos. Por último se exponen las conclusiones derivadas de la experiencia adquirida durante el desarrollo de este Trabajo Especial de Grado, así como se lista el material bibliográfico utilizado para plantear y llegar a la solución propuesta.

CAPÍTULO I – MARCO CONCEPTUAL

En el presente capítulo se definen conceptos fundamentales relacionados con este Trabajo Especial de Grado como son el Reconocimiento Óptico de Caracteres y la computación en la nube ó Cloud Computing. Dentro de este contexto también se describen las tecnologías utilizadas.

El Reconocimiento Óptico de Caracteres es un proceso que se encarga de reconocer y extraer el patrón de texto que se encuentra presente en imágenes digitalizadas. Las herramientas que permiten llevar a cabo dicha labor se conocen como motores OCR.

La computación en la nube es una convergencia de muchos conceptos computacionales que permiten brindar servicios utilizando como medio Internet.

Estos dos conceptos convergen en este T.E.G., donde planteamos un enfoque de computación en la nube para implementar y proveer un servicio de OCR.

1.1 Reconocimiento óptico de caracteres (OCR)

Es llamado OCR (por sus siglas en inglés, Optical Character Recognition) "Reconocimiento Óptico de Caracteres" al reconocimiento de texto que se realiza a través del proceso de traducción mecánica o electrónica de imágenes escaneadas con texto de cualquier tipo, bien sea manuscrito, mecanografiado o impreso permitiendo obtener su representación en texto codificado o cadena de caracteres (ASCII o Unicode) para una máquina como dice Bunke en [1].

Las aplicaciones que se encargan de dicha traducción reconocen símbolos y caracteres pertenecientes a un alfabeto en particular haciendo uso de reconocimiento de patrones, y posteriormente esta información se procesa y queda disponible para su tratamiento posterior en forma digital como por ejemplo un editor de texto.

Como menciona Cheriet en [2], la investigación y desarrollo de OCR comienza en la década de 1950, cuando surge la necesidad de capturar imágenes de texto mediante diversos dispositivos que brindaba la tecnología en ese entonces. Con el pasar del tiempo fue aumentando la necesidad de contar con fuentes estándares que permitiesen agilizar el proceso del OCR, fue entonces cuando en la década de 1960 se crean las fuentes OCR-A (Optical Character Recognition Font A) y OCR-B (Optical Character Recognition Font B) que posteriormente en la década de 1970 fueron adoptadas por la International Standards Organization (ISO), ayudando así a masificar la tecnología de OCR y disminuyendo el tiempo requerido para el procesamiento.

1.1.1 Escenarios para realizar OCR

Existen tres formas de realizar el OCR, como se ve a continuación:

Automático: El OCR se ejecuta automáticamente en el momento que se escanea un nuevo documento, basándose en una configuración previamente establecida.

Manual: El OCR se ejecuta sobre documentos que pueden ya estar previamente escaneados, debiendo previamente seleccionar los archivos sobre los cuales se trabajará. Estos documentos deben contar con una alta calidad de resolución ya que mientras mayor sea la calidad de la imagen se producirán mejores resultados de reconocimiento óptico.

Indexación: En este escenario se facilita la búsqueda de las imágenes relevantes a una búsqueda por texto, para ello se debe contar con imágenes en formato TIFF (Tagged Image File Format) "Formato de fichero de imágenes" o MDI (Microsoft Document Imaging) o "Formato de archivos basado en formato TIFF" que permiten asociarle a una imagen etiquetas empleadas para describir dicha imagen.

1.1.2 Tipos de OCR según la zona de búsqueda

Toda imagen que se procese puede ser indexada por datos presentes en ella, según la ubicación de dichos datos, los distintos sistemas OCR se pueden clasificar en OCR Zonal y OCR Dinámico.

OCR Zonal: Permite leer zonas de texto estáticas en una imagen digitalizada, es decir, los índices o etiquetas deben estar ubicados en cada imagen procesada exactamente en el mismo lugar puesto que el sistema sólo procesará los datos que en esa "zona" se encuentren. El sistema debe proveer un mecanismo que permita establecer la "zona" donde se reconocerán los caracteres delimitando la misma en zonas normalmente rectangulares, esto se hace sobre una imagen de prueba y luego se aplica a todo el conjunto

OCR Dinámico: Permite encontrar los datos importantes para la indexación sin importar que los mismos no estén ubicados en el mismo lugar en cada una de las imágenes procesadas. Por tanto, permite mayor flexibilidad a la hora de reconocer los caracteres relevantes puesto que si no se limita a un espacio estático, cosa que por ejemplo perjudica cuando tenemos el OCR Zonal pues cuando tenemos caracteres en mayúsculas podríamos correr el riesgo de estar posicionados fuera del rango de ubicación de la "zona" y por ende estos podrían ser reconocidos de manera defectuosa.

1.1.3 Etapas del análisis de imágenes de documentos

Como dice Bunke en [1], «todo software OCR emplea un conjunto de métodos que juntos permiten analizar los documentos con los que debe trabajar». Entre ellos tenemos:

- **Adquisición de la imagen:** El primer paso consiste en convertir el documento a una representación numérica, este proceso es llevado a cabo por hardware especializado ya sea un scanner o una cámara. Esta imagen luego pasa a un proceso de binarización en el cual la imagen se transforma a los colores blanco (valor de gris 255) y negro (valor de gris 0) para así diferenciar los símbolos y caracteres de la imagen reduciendo considerablemente la información de la misma y separándola en regiones u objetos de interés en una imagen del resto. El resultado es una imagen binaria segmentada por niveles de gris o de una segmentación por selección de un rango de color determinado y/o en otros casos es el resultado de una selección interactiva de regiones de interés que se utilizarán luego como máscaras de comparación o referencia.
- **Transformación de la imagen**: La imagen recién adquirida pasa por varias etapas para ser mejorada, por lo que en esta etapa se tiene una imagen de entrada y la salida es otra imagen ya transformada. Para esto emplea técnicas como transformación geométrica para corregir la distorsión en la imagen adquirida, filtrado para separar el fondo de la imagen que se quiere procesar, detección de límites y adelgazamiento de la imagen.
- **Segmentación de la Imagen**: En este proceso la imagen se divide en regiones las cuales pueden tener un objeto o un grupo de objetos del mismo tipo. Para lograr esta segmentación se emplean técnicas especializadas para el análisis de imágenes de documentos como:
    a) **Etiquetado de componentes conectados (Connected Component Labeling en inglés)** que consiste en escanear la imagen píxel por píxel de arriba abajo y de derecha a izquierda con la finalidad de identificar las regiones de píxeles conectados a través de la asignación de una etiqueta distintiva para cada componente conectado de la imagen binaria. Estas etiquetas generalmente son números naturales que van desde 1 hasta n, donde n es el total de componentes conectados que posea la imagen de entrada. Un ejemplo de etiquetado de componentes se puede observar en la Figura 2.

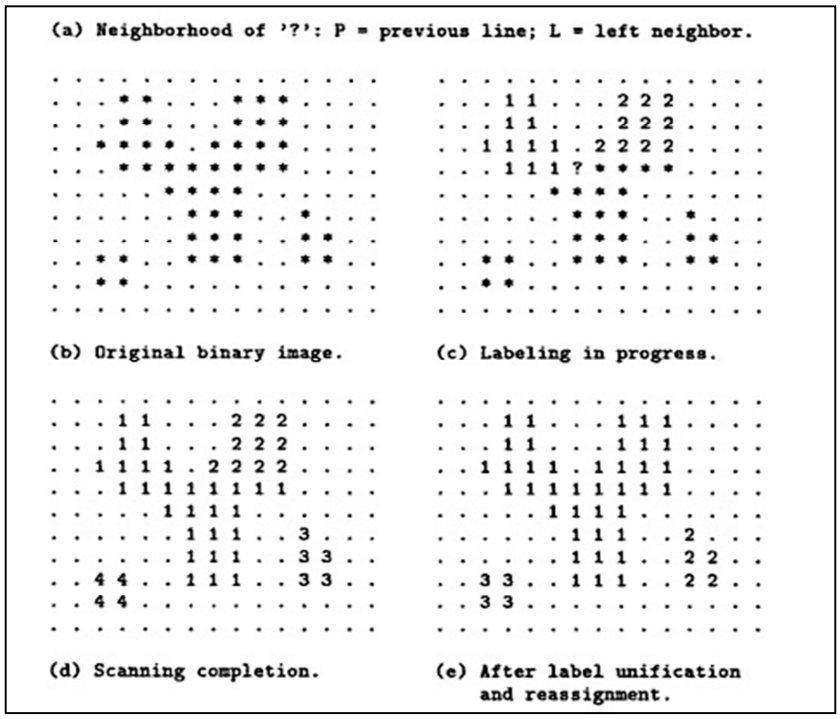

Figura 2. Ejemplo de Etiquetado de Componentes Conectados.

b) **<u>Árbol de descomposición X-Y (X-Y-Tree Descomposition en inglés)</u>** el cual consiste en un algoritmo cuya idea básica es la explotación del hecho de que la mayoría de las imágenes de un documento tienen una estructura vertical y una horizontal de manera que al hacer un análisis de proyección del perfil de una página se segmenta la misma en una sección de bandas blancas(conjunto de columnas contiguas con n píxeles negros) y otra de bandas de texto (conjunto de columnas contiguas con al menos n píxeles si el umbral de comparación es n). Un ejemplo del árbol correspondiente a una imagen es la Figura 3.

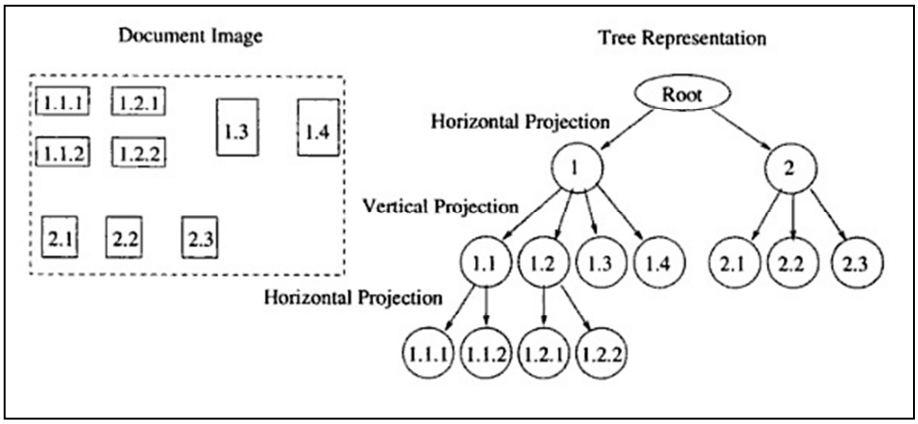

Figura 3. Un documento y su representación en árbol.

c) **RLS (Run-Length Smearing)** es un algoritmo que requiere una proyección de antelación asimétrica. Consideremos una línea que consiste de 0's (píxeles blancos) y 1's (píxeles negros), el RLS primero detecta las series blancas (0's contiguos) y luego convierte las series cuya longitud sea más corta que el umbral T, a series negras. Las series negras permanecen sin ser cambiadas. Por ejemplo, con T=3 el RLS convierte la línea 0 0 0 1 1 0 1 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 en 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1. Para obtener la segmentación, el algoritmo RLS se aplica primero línea a línea y luego columna a columna generando dos bitmaps distintos los cuales son combinados eventualmente por el operador lógico AND. En la Figura 4 se pueden observar las distintas etapas de este algoritmo.

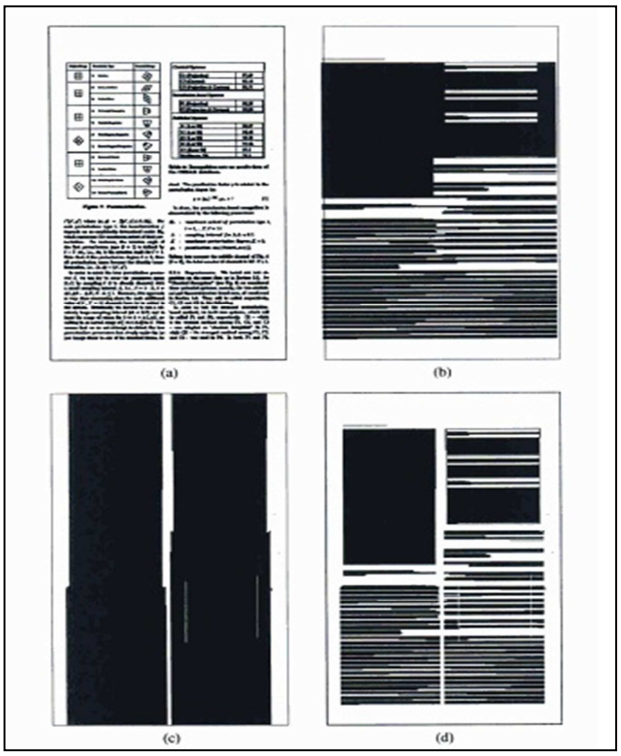

Figura 4. Segmentación de página por el algoritmo RLS.
(a) Página que contiene texto y gráficos. (b) Versión horizontal de (a). (c) Versión vertical de (a). (d) Resultado de la segmentación de (a) por el algoritmo RLS.

d) **La transformada de Hough** es un algoritmo que consiste en que para cada punto que se desea averiguar si es parte de una línea se aplica una operación dentro de cierto rango, con lo que se averiguan las posibles líneas de las que puede ser parte el punto. Esto se continúa para todos los puntos en la imagen, al final se determina qué líneas fueron las que más puntos posibles tuvieron y esas son las

líneas en la imagen. Por ejemplo: Consideremos la ecuación de una línea y= ax + b, en la cual los parámetros (a, b) deben ser determinados, si el punto (x1, y1) pertenece a la línea, entonces es claro que algún par (a, b) que satisfaga la ecuación y1= ax1 + b es solución potencial. En otras palabras, para un punto dado (x1, y1) la curva b= -x1a + y1 en el parámetro de espacio (a, b) describe todas las posibles soluciones. Si n puntos (x1, y1), (x2, y2),….. (xn, yn) se encontraran en la misma línea del espacio de la imagen, sus curvas correspondientes deben interceptarse una a otra en el mismo punto (a*, b*) en el parámetro espacio. En la Figura 5 se ilustra el método.

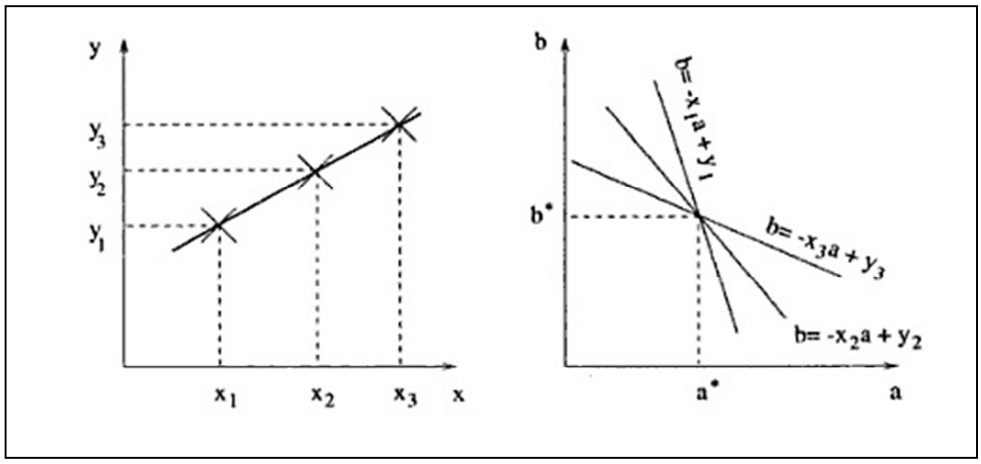

Figura 5. Extracción de una línea por la transformada de Hough.

- **Extracción de características**: Una de las metas del análisis de imágenes de documentos es clasificar caracteres y símbolos en clases, la extracción de características ayuda precisamente a esta clasificación. Aquí es donde se reconocen los patrones de la imagen ya sea empleando enfoques estadísticos o enfoques estructurales.

1.1.4 Utilidad del OCR

Este software es ampliamente usado para digitalizar documentos ya que permite tomar como entrada una imagen y procesar el texto que ésta contiene al igual que cualquier elemento de la misma tal como un dibujo, un esquema, etc., obteniendo por consiguiente un importante ahorro de tiempo ya que evita la transcripción de documentos, ventaja que es bastante notable cuando se digitaliza gran volumen de información.

La principal ventaja es la capacidad de buscar contenido dentro de un documento escaneado sin OCR. Esto supone realizar búsquedas rápidas sin tener que perder tiempo buscando entre todo el documento, página a página, palabra a palabra, para encontrar algo concreto.

Además, este tipo de soluciones en organizaciones que ya tienen hardware de escaneo (equipos multifunción o escáneres) supone el que no haya que reemplazar estos equipos por escáneres más modernos, en muchos casos con la misma calidad de escaneo, y con el único aporte del software OCR en el dispositivo.

Dentro de un Software de Gestión Documental, el Software OCR puede realizar búsquedas directamente sobre los archivos en formato de imagen (por ejemplo una imagen en formato JPG) que contienen texto, y sólo se utiliza este software en una única ubicación, el servidor que alberga el Software de Gestión Documental.

1.1.5 Software Tesseract

Según se menciona en [3], Tesseract es un motor de software libre para el Reconocimiento Óptico de Caracteres compatible con varios Sistemas Operativos. Fue originalmente desarrollado como software propietario por Hewlett-Packard entre 1985 y 1995. En 2005, Hewlett-Packart en conjunto con UNLV lanzó una versión de Tesseract Open Source. El desarrollo de Tesseract es patrocinado actualmente por Google y trabaja bajo Licencia Apache versión 2.0. Tesseract es considerado uno de los motores OCR libres más preciso en reconocimiento óptico de caracteres de los actualmente disponibles en el mercado.

Tesseract está disponible para Windows, Linux y Mac OS X, sin embargo sólo ha sido rigurosamente probado por sus desarrolladores en Windows y la distribución de Linux de Ubuntu.

La primera versión de Tesseract incluyendo la versión 2 sólo podía aceptar con imágenes con una columna de texto simple como entradas. Estas primeras versiones no incluían análisis de estructura del documento para reconocer texto de varias columnas, imágenes y ecuaciones producidas por salidas ilegibles. Desde la versión 3.00, Tesseract ha soportado formato de texto de salida, información posicional hOCR y análisis de diseño por página. El soporte a una serie de nuevos formatos de imagen fue agregado usando la librería de Leptonica. De manera que Tesseract puede detectar si el texto es monoespaciado o proporcional.

Las primeras versiones de Tesseract podían reconocer texto en inglés solamente. A partir de la versión 2 Tesseract era capaz de procesar inglés, italiano, francés, alemán, brasileño portugués, español y holandés. A partir de la versión 3 podía procesar Arábigo, Búlgaro, catalán, Chino

(simplificado y tradicional), danés (secuencia de comandos estándar y Fraktur), alemán, griego, finlandés, francés, hebreo, croata, húngaro, indonesio, italiano, japonés, coreano, letón, lituano, holandés, noruego, polaco, portugués, rumano, ruso, eslovaco (estándar y Fraktur), esloveno, español, serbio, sueco, tagalo, tailandés, turco, ucraniano y vietnamita. Tesseract puede cualificarse para trabajar en otros idiomas también.

¿Cómo funciona Tesseract OCR?

Como dice en [4], Tesseract funciona por terminal (aunque es posible encontrar algún GUI en Java como por ejemplo: jtOCR). Para invocarlo es necesario contar con una imagen en algún formato común como JPG, TIFF, entre otros. Para Instalar Tesseract (inglés y español) en distribuciones basadas en Debian, se procede con el siguiente comando:

$ sudo apt-get install tesseract-ocr-spa tesseract-ocr-eng

Esto instalará Tesseract con el soporte para español e inglés. Es necesario instalar inglés si no va a definirse el parámetro de lenguaje. Si la distribución de Linux a utilizar no está basada en Debian se puede compilar el código fuente:

$ wget -c http://tesseract-ocr.googlecode.com/files/tesseract-2.04.tar.gz

$ tar xzvf tesseract-2.04.tar.gz

$ cd tesseract-2.04

$ ./configure

$ make

$ sudo make install

Para aplicar el OCR sobre los TIFF debe ejecutarse la siguiente línea de comando:

$ tesseract archivo.tif archivo.txt -l spa

El "-l spa" le dice a Tesseract que el texto está en Español donde "archivo.txt" es el resultado en texto plano del origen. La calidad del resultado depende íntegramente de la calidad de la imagen.

Interfaces de usuario

En la Figura 6 se observa un interfaz de usuario desarrollada para emplear el motor Tesseract de manera más intuitiva que por la línea de comandos.

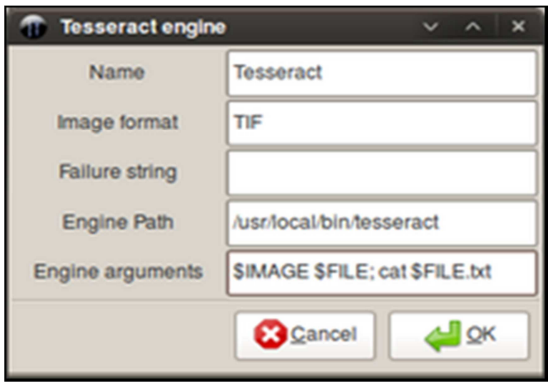

Figura 6. Interfaz de Tesseract Software

Como se menciona en [3] Tesseract no viene con una GUI y corre desde una consola de línea de comandos. Existen varios proyectos separados que proveen una GUI para Tesseract.

- FreeOCR. Una GUI de Tesseract para Windows.
- gImageReader. GTK GUI Frontend para Tesseract que soporta columnas seleccionadas y partes del documento, puede abrir archivos e imágenes multipáginas PDF, soporta todos los formatos, puede transmitir un área seleccionada a Tesseract para el reconocimiento y corrección ortográfica de la salida.
- Gscan2pdf. Es una GUI para producir PDF y DJVus de documentos escaneados.
- OCRFeeder. Caracteriza una interfaz de usuario grafica GTK muy completa que permite a los usuarios corregir detalles de caracteres no corregidos, establecer estilos de párrafo, limpiar imágenes de entrada, importar PDFs, guardar y cargar el proyecto, exportar todo a múltiples formatos, etc.
- OcrGUI. Es una GUI de Linux escrita en lenguaje C y que usa Glib y GTK como frameworks, además soporta Tesseract y GOCR. Incluye corrector ortográfico usando Hunspell, un chequeador ortográfico open Source.
- Qiqqa. Es una herramienta de gestión referencial de PDF gratuita que utiliza Tesseract para interpretar PDFs escaneados para búsquedas full-índex.
- Tesseract GUI. Una GUI de software libre para Mac OS X.
- TextRipper. Una GUI de Tesseract y Ocrad para Linux con múltiples-paginas, múltiples-columnas y soporte de selección de archivos.

1.1.6 Ruby on Rails

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993,

y lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU. Ruby es un lenguaje de programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre. Ruby ha sido descrito como un lenguaje de programación multiparadigma: permite programación procedural (definiendo funciones y variables fuera de las clases haciéndolas parte del objeto raíz Object), con orientación a objetos, (todo es un objeto) o funcionalmente (tiene funciones anónimas, clausuras o closures, y continuaciones; todas las sentencias tienen valores y las funciones devuelven la última evaluación). Soporta introspección, reflexión y metaprogramación, además de soporte para hilos de ejecución gestionados por el intérprete. Ruby tiene tipado dinámico y soporta polimorfismo de tipos (permite tratar a subclases utilizando la interfaz de la clase padre). Ruby no requiere de polimorfismo de funciones al no ser fuertemente tipado (los parámetros pasados a un método pueden ser de distinta clase en cada llamada a dicho método).). La Figura 7 presenta la arquitectura de Ruby on Rails.

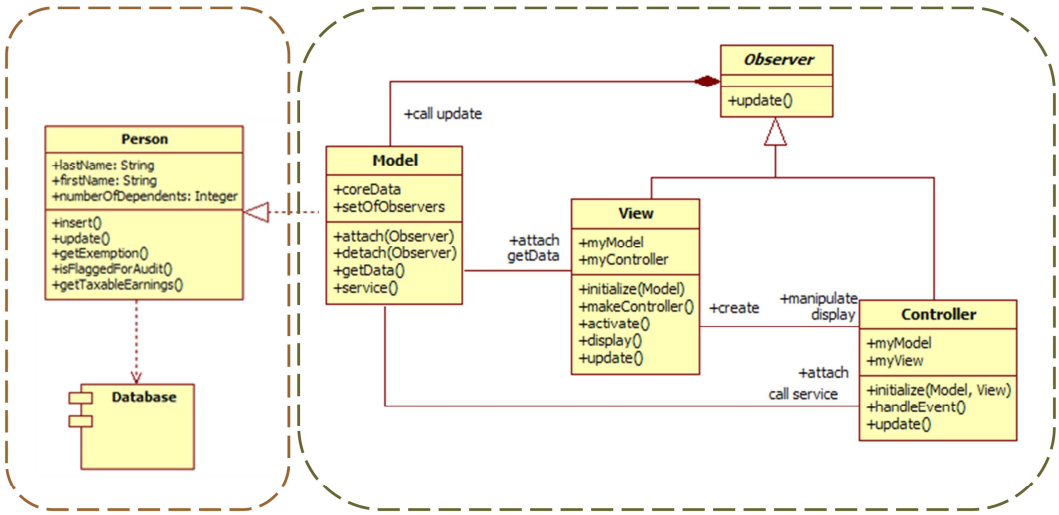

Figura. **Active Record**
Fuente: Fowler (2005)

Figura. **Modelo-Vista-Controlador**
Fuente: Trygve Reenskaug (1979: Buschmann, 2001)

Accede a la data de manera persistente

Separa la data (modelo) de la IU (vista)

Figura 7. Arquitectura de Ruby on Rails.

1.2 Computación en la nube (Cloud computing)

Como lo dice Ambrust en [5], Cloud Computing se refiere, por una parte, a las aplicaciones entregadas como servicios sobre internet y por otra parte al hardware en Datacenters que proveen estos servicios. Bennett y Bhuller en [6] mencionan que Cloud Computing es la convergencia y evolución de muchos conceptos de virtualización, de aplicaciones distribuidas, de Grids que plantean un enfoque flexible para el despliegue y escalado de aplicaciones tal y como lo menciona Kajeepeta en [7]. Como lo dice Boss en [8], Cloud Computing es un término utilizado para describir una plataforma y el tipo de aplicación, siendo una plataforma que dinámicamente aprovisiona, configura y reconfigura servidores como sea necesario.

Esta sección se enfoca en describir computación en la nube como una red de servicios, teniendo en cuenta aspectos como el diseño, los modelos de red y los protocolos de comunicación. La infraestructura física se implementa generalmente a través de Datacenters que según el tipo podrían ser Nubes Públicas o Privadas. También se utiliza en gran parte la virtualización debido a la escalabilidad y disponibilidad que presenta.

1.2.1 Evolución de la computación en la nube

El término computación en la nube no es nuevo, ni revolucionario, así lo afirma Quan en [9]. Empezó en los años 1980's bajo conceptos de Grid Computing, aunque con ciertas diferencias y enfatizado a servidores virtuales; luego en los años 1990's se expandió el concepto de virtualización elevando el nivel de abstracción de los servidores virtuales, primero como plataforma virtual y luego como aplicaciones virtuales; más adelante se conoció el término Utility Computing, que ofrece clusters como plataformas virtuales; recientemente el término software como servicio (SaaS) elevando el nivel de virtualización a las aplicaciones, con un modelo de negocio no recargado en recursos consumidos.

El concepto de computación en la nube combina los términos anteriores de Grid, Utility y SaaS, siendo un modelo emergente en donde los usuarios pueden tener acceso a las aplicaciones desde cualquier lugar a través de dispositivos conectados.

1.2.2 Escenarios de implementación

En [10] se proponen tres escenarios para la formación de la tecnología de computación en la nube, a saber, Nube Privada, Nube Pública y Nube Híbrida. Estos escenarios se han convertido en medios atractivos para el intercambio computacional, de almacenamiento y de recursos de red

entre desarrolladores de servicios múltiples y de aplicaciones de prestación de servicios. No hay que olvidar la capacidad de reasignar dinámicamente los recursos utilizando tecnologías de virtualización, ayudando a mitigar la necesidad de inversiones adicionales en infraestructura en tiempos de alta demanda.

Nube privada

Las Nubes Privadas o Internal Cloud, tal como dice Lasica en [10] son escenarios donde las empresas o particulares realizan sus operaciones fuera de línea, ejecutando aplicaciones seguras en Datacenters.

Lasica en [10], menciona que Internal Cloud aplica los conceptos de computación en la nube a recursos propios de la empresa o un particular que consume el servicio, proveyendo la capacidad de manejar aplicaciones web nuevas y existentes, mientras se provee de seguridad y regulación.

Nube pública

Las Nubes públicas o External Cloud son escenarios donde las empresas o particulares necesitan mover datos o aplicaciones desde su interior al exterior. Este escenario público se conecta con otros escenarios. External Cloud involucra recursos y servicios IT que son vendidos , tales como autoservicio, aprovisionamiento en demanda y pago por utilización, todos estos servicios accedidos a través de navegadores web o a través de API's.

Nube híbrida

Nube Híbrida o Inter Cloud. Lasica en [10] hace referencia a una mezcla entre los dos escenarios anteriores. Es un tipo de escenario semipúblico, el cual se comporta como una Nube Privada con la particularidad de que ciertas organizaciones pueden compartir su información con ciertos niveles de permiso, por ello el término semipúblico.

El control de la Nube Pública lo hace el proveedor, mientras que el control de la Nube Privada lo hace la organización, y la finalidad es que a través de ambos enfoques se puedan satisfacer las necesidades de un sistema.

Inter Cloud como escenario, brindaría la capacidad de elegir los proveedores de servicio, y los proveedores de servicio federados serían capaces de compartir las cargas de servicio, siendo una relación más flexible con respecto a las nubes privadas y las nubes públicas.

1.2.3 Niveles de servicio

Los diferentes niveles de servicio que componen la computación en la nube son:

- **Infrastructure as a Service (IaaS)** o infraestructura como servicio como dicen Olhman y Eriksson en [12] es un servicio que entrega tanto hardware como software tal como un servicio. El ejemplo más común es el hosting, el cual, nos provee de hardware como un servidor y de software como un webserver, sin embargo, este concepto ha evolucionado a infraestructura como EC2 y S3.

Otra manera de ver IaaS es como la manera de compartir recursos para ejecutar servicios, típicamente utilizando tecnología de virtualización en donde múltiples usuarios utilizan dichos recursos. Los recursos pueden fácilmente escalar cuando la demanda se incrementa, y generalmente se utilizan métodos como pago por uso.

Uno de los servicios que toma importancia en la infraestructura es el Cloud Storage que se menciona a continuación:

Cloud Storage. Es almacenamiento localizado ya sea en Datacenters públicos o privados, separados del almacenamiento primario. Según mencionan Kamaraju y Nicolas en [13], pertenece al nivel de infraestructura como servicio, la manera de implementación es a través del Service Oriented Architecture (SOA), y la localización tiene diferentes variaciones, Cloud Storage puede ubicarse en:

- Un Datacenter público,
- Un Datacenter privado, o
- Separado del almacenamiento primario.

La manera de acceso puede ser de dos formas: Directamente como bloques o archivos; o indirectamente a través de aplicaciones que están ubicadas en el mismo lugar del almacenamiento. Según Mendoza en [14] hay dos tecnologías que proporcionan métodos de almacenamiento: La primera es Storage Area Network (SAN) que son switches de redes de alta velocidad que permiten que múltiples computadoras tengan acceso compartido a varios dispositivos de almacenamiento; y la segunda es Network-Attached Storage (NAS) que vienen como aplicaciones NAS o Gateways NAS, son servidores de archivos virtuales que tienen soporte a protocolos como NFS, siendo un dispositivo que directamente concede a la red y que tiene capacidades de compartir archivos.

Los protocolos utilizados para Cloud Storage son SOAP o REST, más adelante se detallarán dichos protocolos. También existen ciertas barreras para su adopción, tales como, el ahorro de

costos no es significante, se pone en riesgo la privacidad de los datos, cuestiones de migración, disponibilidad de datos y contratos como SLAs.

- **Platform as a Service (PaaS)** o plataforma como servicio, es entregar una plataforma de desarrollo de aplicaciones como un servicio para desarrolladores en la web. Generalmente se provee de herramientas tipo middleware, por ejemplo, Google AppEngine. Además de dicha entrega, también se ofrece un ambiente de ejecución como el servidor de aplicaciones.
- **Software as a Service (SaaS)** o software como servicio provee la administración y hosting de aplicaciones con sus propios Datacenters, se maneja el término de múltiples inquilinos, por ejemplo Oracle CRM On Demand o Salesforce.

Según menciona Dver en [15] se compara a SaaS con la controversia que generó la computación cliente-servidor para reemplazar la computación de mainframes, y para el usuario final SaaS es un simple concepto, el usuario solamente ingresa a una aplicación a través del navegador web sin saber en dónde se aloja o como está siendo servida.

En [11], Mikkilineni y Sarathy mencionan otro nivel de servicio, aunque ciertos autores prefieren alojarlo entre los mismos tres niveles de servicio y no crear uno nuevo, es el llamado ITaaS, IT como servicio, siendo un modelo de servicio donde una organización o individuo contrata con un proveedor de servicios para obtener conectividad de red y cualquier otro servicio incluido, como backup de red, recuperación de desastres, VPN, conferencias web, etc. Este nivel es muy general y puede abarcar a los tres anteriores pero de una forma unificada. En la Figura 8 se muestra gráficamente este concepto de brindar hosting web bajo el enfoque de la computación en la nube.

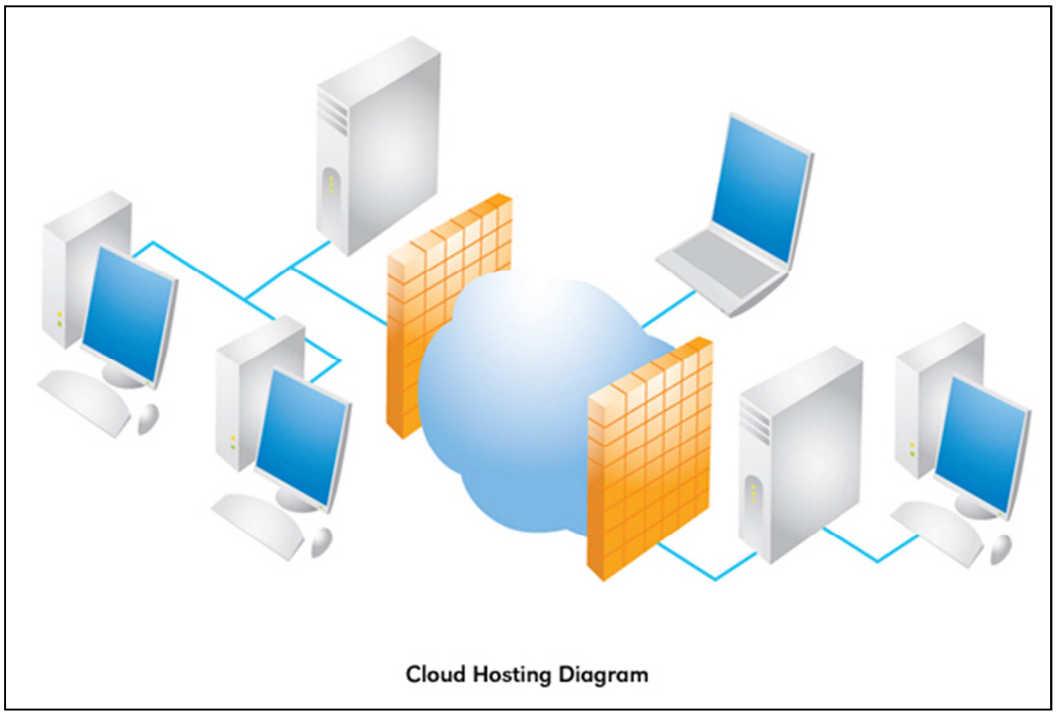

Figura 8. Diagrama Cloud Hosting.

1.2.4 Plataformas de trabajo para computación en la nube

A continuación se describe una serie de plataformas que potencian los sistemas en la nube sobre Ruby, todas ellas brindan una opción que se debe considerar cuando se pretende desarrollar un sistema en nube. Abarcaremos los puntos más resaltantes de cada una de ellas, en particular haremos énfasis en la plataforma conocida como CloudCrowd, desarrollando los conceptos claves de la misma, debido a su aporte significativo para el presente trabajo. No se pretende hacer un análisis comparativo entre todas las aquí mencionadas, ni tampoco se garantiza que son las mejores opciones ni mucho menos las únicas que se pueden tener en cuenta.

RightScale

RightScale como se menciona en [16] es una plataforma que permite gestionar la infraestructura en la nube (IaaS) de múltiples proveedores. RightScale Gems son un conjunto de gemas de software libre que proveen a Ruby de interfaces para muchas plataformas de computación en nube.

Entre las plataformas a las que se les brinda soporte están:

- **Amazon Web Services:** Como se menciona en [17] es un conjunto de servicios que ofrece Amazon.com y que en conjunto forman una plataforma para computación en la nube. Estos servicios pueden ser empleados de manera independiente o conjunta como se dice en [18]. Entre los servicios destacan Amazon EC2 y Amazon S3, un servicio web que configura la capacidad dinámicamente según los requerimientos y un servicio web de almacenamiento en internet respectivamente.
- **GoGrid:** Tal y como se menciona en [19] es un proveedor de IaaS que sirve de hosting en la nube a máquinas bajo Linux y Windows. Brinda servidores tanto físicos como virtuales, almacenamiento, balanceador de carga y seguridad.
- **SliceHost:** Proveedor de servidores virtuales, que cuenta con herramientas de respaldo, panel de control para dispositivos móviles, acceso por consola al servidor, entre otras [20]. Emplea a su vez el software de virtualización Xen Hypervisor, que permite entre otras cosas, ejecutar múltiples sistemas operativos concurrentemente.
- **Flexiscale:** Es una plataforma que permite a los clientes iniciar, detener y configurar servidores a medida de sus necesidades. Fue la segunda plataforma para la nube en Europa, así se menciona en [21]. También emplea el Xen Hypervisor y una Red de Área de Almacenamiento (SAN por sus siglas en inglés).

CloudCrowd

CloudCrowd es un servicio que tiene como intención hacer el procesamiento distribuido fácil para los programadores de Ruby. Algunos trabajos que serían apropiados para CloudCrowd son los siguientes:

- Generar o cambiar el tamaño de las imágenes.
- Ejecutar la extracción de texto OCR o en archivos PDF.
- Realizar codificación de vídeo.
- Poder migrar un conjunto de archivos de gran tamaño o base de datos.
- Raspado Web.

El servidor central mantiene un historial del estatus de todos los trabajos y unidades de trabajo activas, pero no ejecuta ninguna acción de procesamiento por sí mismo. Cada Split, proceso y merge es distribuido a un nodo y corre en un proceso trabajador o worker process. Los

workers corren una unidad de trabajo única, retornan sus resultados, limpian o resetean sus archivos temporales y luego expiran inmediatamente. De esta manera, cuando la cola de trabajo está vacía todos los nodos lo están también, y las tareas a ejecutar no generan pérdida de memoria y tiempo en ningún momento.

Tal y como se menciona en [22], una causa común del embotellamiento en sistemas de procesamiento distribuido es el almacenamiento de archivos, el cual es usado como cámara de resultados intermedios, así como para entradas y salidas. Por ejemplo, el MapReduce original de Google usa un sistema de archivos distribuido en una cámara, Hadoop usa el tradicional sistema de archivos HDFS Hadoop. CloudCrowd usa S3, lo que hace una idea especialmente buena para desplegar el clúster CloudCrowd en EC2. El soporte de almacenamiento alternativo puede ser conectado a la clase AssetStore y seleccionado en config.yml.

Estas limitaciones hacen a CloudCrowd apropiado para volúmenes moderados de un trabajo muy costoso (tanto para el CPU, la memoria o ancho de banda). El método de división le permite convertir un solo archivo grande en una lista muy paralela de las unidades de trabajo, y los nodos funcionarán hasta max_workers (número máximo de workers proccess) por nodo para manejar el trabajo. En la Figura 9 podemos observar la anatomía de la plataforma Cloud Crowd.

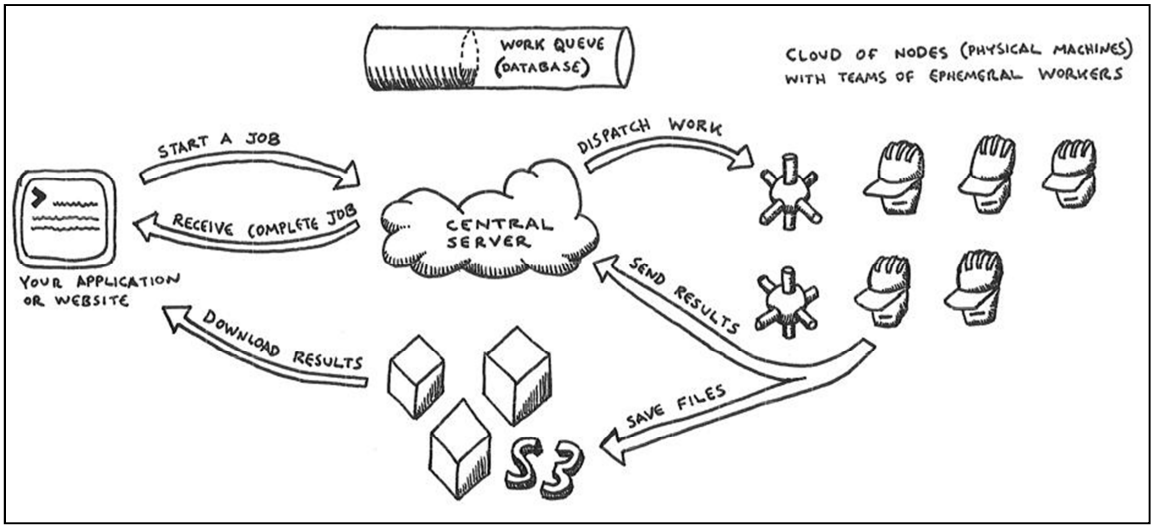

Figura 9. Anatomía de un Cluster CloudCrowd.

CloudCrowd puede compartir código con aplicaciones en Rails, tomando ventaja de los modelos pre-existentes y desarrollando operaciones costosas de base de datos en forma paralela. La técnica precisa para llevar a cabo esta tarea puede variar ya que no todas las aplicaciones necesitarán cargar toda la pila de Rails con la finalidad de poder realizar o ejecutar una tarea. En DocumentCloud, se almacenan las acciones en app/actions, y se carga el ambiente Rails a fin de

acceder sin problemas a todos los modelos de la aplicación, al igual que durante una solicitud web real porque CloudCrowd llama a un worker por cada unidad de trabajo, Rails se carga sólo una vez, cuando un nodo gira y los procesos hijos no incurren en gastos de carga del mismo.

He aquí un fragmento de ejemplo de código en la Figura 10, que se usa para instalar el entorno de Rails para una acción. El CloudCrowd.node poco se asegura de que la pila de Rails no siempre se cargue en el servidor central.

```
# Inherit Rails environment from Sinatra.
RAILS_ROOT = File.expand_path(Dir.pwd)
RAILS_ENV = ENV['RAILS_ENV'] = ENV['RACK_ENV']

# Load the DocumentCloud environment if we're in a Node context.
if CloudCrowd.node?
  require 'rubygems'
  require 'activerecord'
  ActiveRecord::Base.logger = Logger.new(STDOUT)
  require 'config/environment'
end
```

Figura 10. Fragmento de Código en Ruby on Rails.

Heroku

Como se dice en [23], Heroku es una plataforma para aplicaciones Ruby en la nube, fundada por Orion Henry, James Lindenbaum y Adam Wiggins en el año 2007. Este sistema enfoca su filosofía en desprender a los desarrolladores de las labores de buscar, analizar y configurar los entornos en los que sus aplicaciones correrán, dejando de esta manera que se puedan enfocar exclusivamente en el desarrollo de sus productos.

Este sistema se define como un proveedor de plataformas en forma de servicio (PaaS por las siglas de Platform as a Service) el cual le brinda al desarrollador las herramientas necesarias para construir, desplegar y ejecutar efectivamente aplicaciones Ruby en la nube. Entre los puntos más atractivos de su arquitectura se encuentra la facilidad de desplegar una aplicación, ya que gracias a su integración con la herramienta de control de versiones Git, el proceso de subir el código nuevo a la plataforma se reduce a añadir el repositorio remoto de Heroku en el proyecto mediante '*git remote add heroku git@heroku.com:appname.git*', y así, una vez configurado, el código puede ser actualizado ejecutando *git push heroku master.*

Aparte de la facilidad para actualizar el código de la aplicación, está la capacidad autónoma de Heroku para administrar y renovar los procesos (llamados Dynos) que usa la aplicación en su

ejecución, mediante un elemento clave llamado Dyno Manifold. Inclusive, cada vez que se despliega una nueva versión del código de la aplicación, o existen cambios de configuración, los dynos son reiniciados debidamente para completar el despliegue. El Dyno Manifold también se encarga de detectar Dynos que estén actuando de manera anormal, reiniciándolos automáticamente e incluso forzándolos a un descanso prolongado si continúan en estado problemático. Aparte de estas acciones, el administrador también puede administrar los Dynos manualmente.

Más acerca de los Dynos, estos procesos tienen características particulares como su capacidad de ser aumentados/decrementados de manera transparente en la aplicación, son monitoreados constantemente y su ejecución se lleva en instancias físicas distintas, por lo que se asegura la disponibilidad de recursos, se obtiene un sistema más robusto y se garantiza una redundancia óptima. Adicionalmente, los Dynos se pueden configurar para cumplir tareas de procesamiento web, procesamiento de tareas internas o incluso para tareas programadas.

También se debe destacar la facilidad para crear configuraciones dedicadas dependiendo del despliegue a realizar, ya sea desarrollo, pruebas o producción. Permiten el uso de variables de configuración que pueden cambiar dependiendo del tipo de despliegue que se está realizando. Esto incluye las credenciales de acceso a cualquier servicio que pueda usar una aplicación. Esta configuración, las labores de despliegue y el monitoreo de la plataforma se puede realizar tanto por una interfaz de línea de comandos (CLI, siglas de Command-line Interface), por una consola web o usando un API específico como se menciona en [24].

Respecto al monitoreo, en [25] se menciona que Heroku ofrece un componente llamado Logplex, el cual permite observar a un nivel bastante específico todas las operaciones que está ejecutando la aplicación, desde ver como maneja las peticiones HTTP hasta como se administran los Dynos además de su status y cómo transcurre el despliegue del código nuevo.

Otro de los puntos clave de Heroku es el automantenimiento de la plataforma, lo que permite al desarrollador evitar hacer el mantenimiento rutinario de un sistema, como la actualización del software usado, actualizaciones al sistema operativo, barrido de logs al llenar el disco, mantenimiento de hardware, entre otras tareas que al no ser cumplidas causan un efecto de erosión del software. Gracias a que el servicio de Heroku libera al desarrollador de esas tareas, estos pueden dedicarle más tiempo a tareas directas relacionadas con el crecimiento de su sistema y mejoras en él. Para el mantenimiento propio de la aplicación, actualizaciones y otras

labores, la plataforma cuenta con un modo de mantenimiento, el cual redirige las peticiones HTTP a una página estática configurable para evitar tráfico mientras se completa el mantenimiento.

En el aspecto de la migración de datos, Heroku utiliza una herramienta llamada Taps para importar/exportar la información de una base de datos. En [25] se menciona que Taps es una herramienta implementada en Ruby que mediante la creación de una instancia en el servidor indicando la dirección, usuario y contraseña de la base de datos a extraer y otra instancia en el servidor a obtener la base de datos, se encarga de copiar el esquema y la información de un equipo a otro, evitando la tarea de hacer un dump y enviarlo al servidor destino. Además una de las características más importantes de esta utilidad es que se puede copiar información entre sistemas de base de datos distintos (por ejemplo, de MySQL a PostgreSQL) y ya que Heroku tiene esta herramienta por defecto en su plataforma, facilita en gran medida la carga y recuperación de información.

CAPÍTULO II – MARCO APLICATIVO

En el presente capítulo se detalla el desarrollo de la aplicación en la nube para el procesamiento OCR, así como su justificación ante la problemática existente. Se presentan los dos escenarios, el actual y el desarrollado en este trabajo, de manera que se pueda apreciar la ventaja de la solución de computación en la nube para este tipo de procesamiento respecto al proceso actualmente empleado dentro del sistema de Documentos Interactivos del Banco Central de Venezuela. Sobre la aplicación aquí desarrollada se especifican sus componentes y la integración de los mismos, igualmente en el capítulo se podrán observar los resultados obtenidos de las pruebas realizadas con ambos enfoques.

2.1 Situación actual

Actualmente los sistemas de Reconocimiento Óptico de Caracteres (OCR) resultan de vital importancia para proyectos donde se cuenta con una gran cantidad de información no digitalizada que debe ser servida al público para su visualización, edición y manipulación. Las instituciones que se dedican a brindar este tipo de servicios se enfrentan a sistemas centralizados que no siempre emplean de la mejor manera los recursos computacionales disponibles dentro de la organización. Agregado a esto, tienen dificultad para migrar proyectos existentes a plataformas más novedosas debido a la poca capacidad para acoplar nuevas funcionalidades a los sistemas.

La Facultad de Ciencias de la Universidad Central de Venezuela desarrolló un sistema para el Banco Central de Venezuela [26], mediante el cual se digitalizan distintos documentos de interés público y se hacen disponibles para su consulta mediante un portal web. Esta aplicación cuenta con la arquitectura mostrada en la Figura 11, donde se distinguen los dos componentes del sistema, la aplicación de administración para controlar el proceso de digitalización e indexación de documentos, y el portal de publicación propiamente dicho.

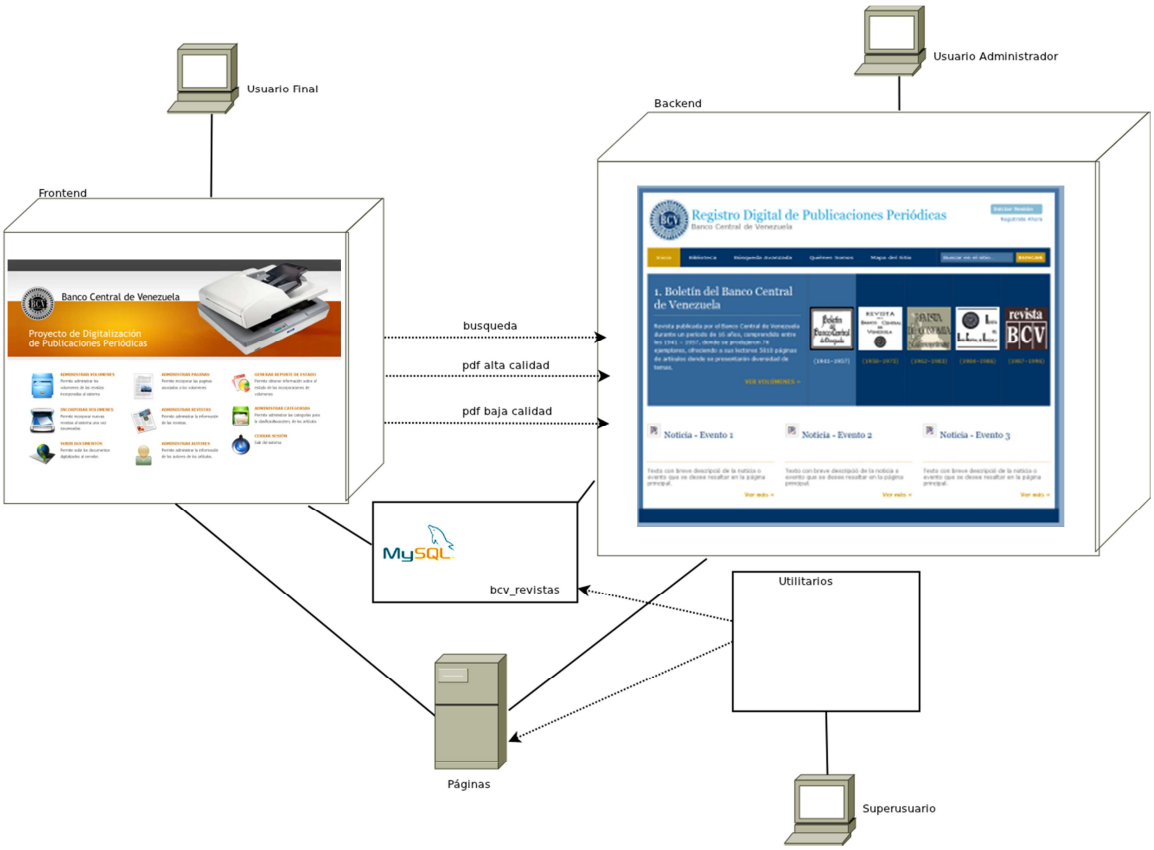

Figura 11. ***Arquitectura del Sistema actual del BCV***

En la Figura 12, se refina la arquitectura del sistema presentando el detalle de las tecnologías utilizadas, así como la conexión entre ellas.

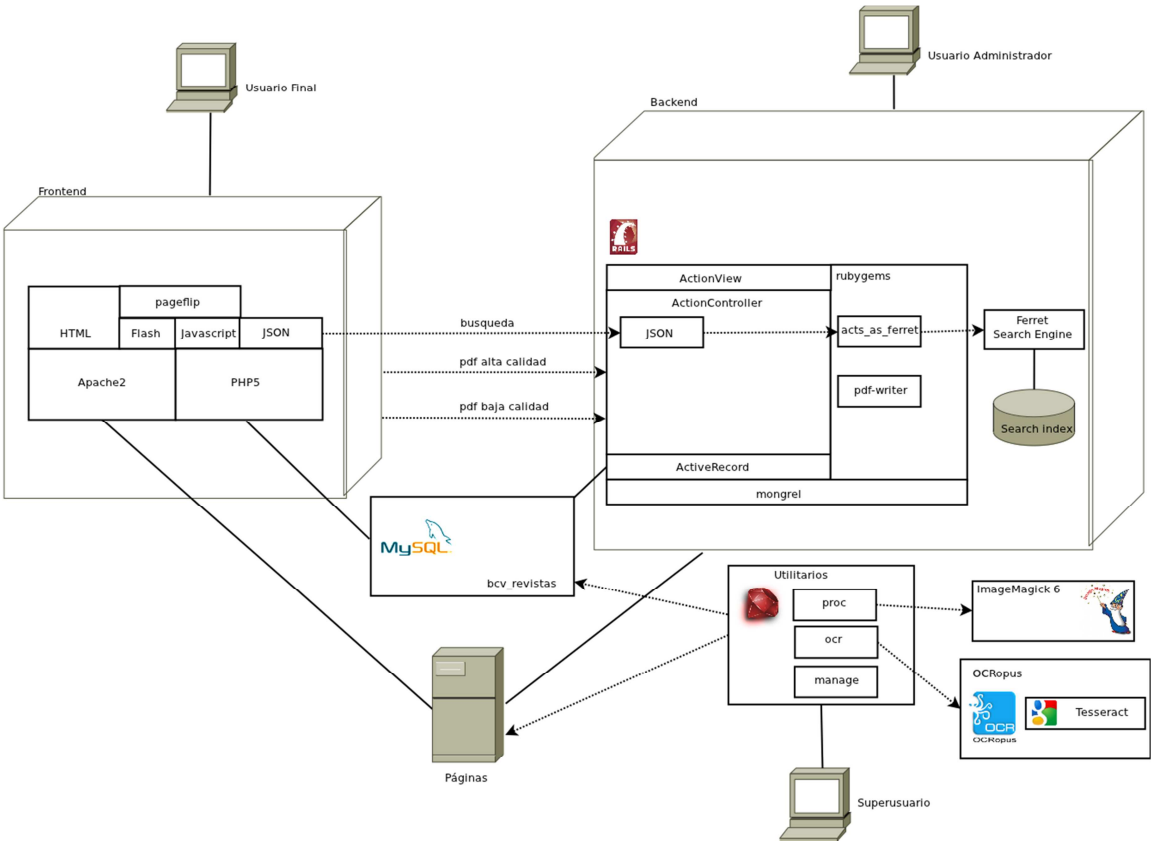

Figura 12. ***Arquitectura del Sistema Actual del BCV.***

Este sistema, a pesar de cumplir con su objetivo, requiere mejorar su rendimiento para procesar el alto volumen de documentos que son incorporados al sistema. Por otra parte, es necesario mejorar los resultados de la digitalización, en cuanto al reconocimiento de los caracteres presentes en las publicaciones tratadas.

Como se observa en la Figura 12, se emplea la herramienta OCRopus la cual hace uso del motor Tesseract descrito en la sección 1.1.5 de este documento. La versión utilizada de este motor es antigua, por lo que los resultados no son los mejores en su reconocimiento, resultando en perdida de precisión en las búsquedas posteriores realizadas por el usuario. El procesamiento del OCR en el sistema actual es secuencial, es decir, se procesa en el servidor una página a la vez, desaprovechando el hardware con que se cuenta para la operación de esta aplicación.

Dentro de la Figura 12 podemos observar que el sistema cuenta con algunos componentes catalogados como utilitarios, entre los cuales se encuentra el llamado OCR, encargado de realizar el procesamiento OCR de los documentos para el sistema. Este utilitario no se encuentra dentro de la aplicación web administrativa del sistema de Documentos Interactivos BCV, por lo que su forma de uso no es familiar a la del resto del sistema.

2.2 Solución propuesta

A partir de la situación antes expuesta, se propone tomar como caso de estudio el sistema de digitalización de documentos de la aplicación de Documentos Interactivos BCV desarrollado en la Facultad de Ciencias, y explorar el impacto de introducir nuevas tecnologías para implementar el proceso de extracción de caracteres. Específicamente se propone utilizar una arquitectura de computación en la nube para mejorar el tiempo requerido para esta tarea particular del sistema.

A su vez, se propone mejorar la calidad de los resultados obtenidos en el proceso de reconocimiento de caracteres en los documentos, mediante la utilización de una versión más actual del motor OCR que se emplea.

El servicio que se desarrollará en este T.E.G. tiene como arquitectura la mostrada en la Figura 13.

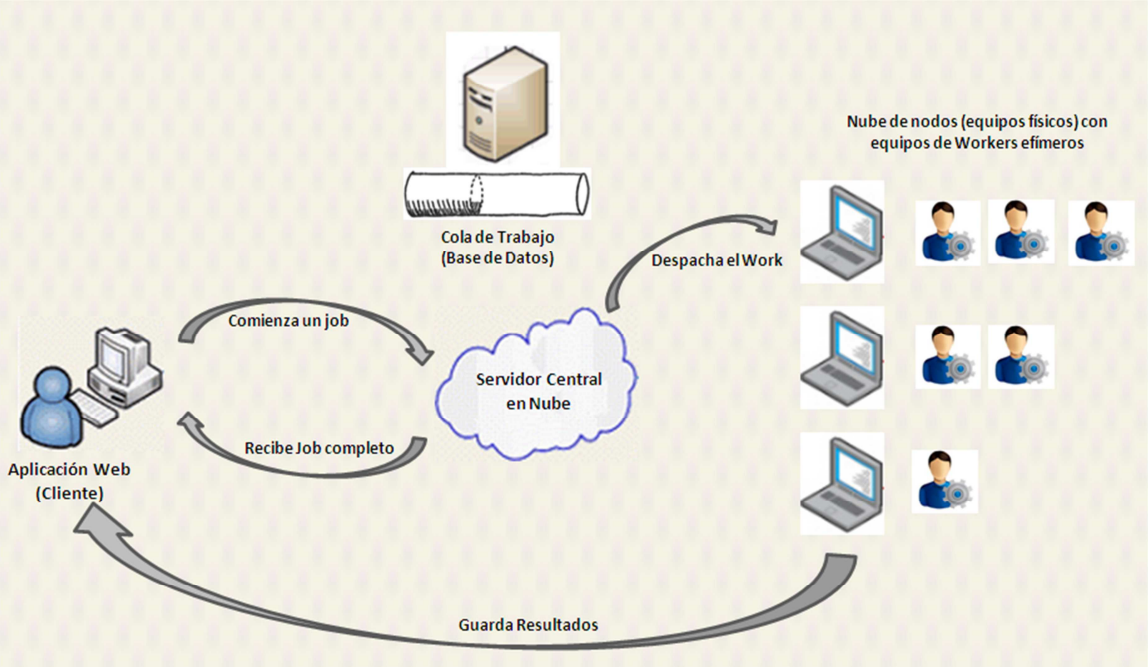

Figura 13. Arquitectura del Sistema en Nube.

Bajo esta arquitectura se pretende que el procesamiento del OCR del sistema sea llevado a cabo por un servicio en nube que emplea la plataforma Cloud Crowd, la cual fue descrita en este documento en la sección 1.2.4; lo cual permite usar de manera eficiente los recursos del servidor y contar con el uso de una versión más actualizada del motor OCR Tesseract para obtener mejor prestaciones respecto a la precisión del OCR. Este servicio está integrado a la aplicación de

administración del sistema de Documentos Interactivos BCV actual mediante un vínculo que permita procesar el OCR de los documentos, sustituyendo así el utilitario externo que se tiene actualmente.

2.3 Objetivos

2.3.1 Objetivo general

El objetivo de este Trabajo Especial de Grado es desarrollar una aplicación para el reconocimiento óptico de caracteres empleando técnicas de programación bajo el enfoque de computación en la nube, que pueda ser integrada al proyecto Documentos Interactivos BCV.

2.3.2 Objetivos específicos

- Desarrollar una aplicación como servicio OCR adaptada a los requerimientos del proyecto "Documentos Interactivos BCV".
- Integrar la aplicación OCR bajo Cloud Computing a la aplicación web existente para administración del proyecto Documentos Interactivos del BCV.
- Evaluar el tiempo de procesamiento por documento enviado al servicio desarrollado.
- Medir la precisión del motor OCR en base a los resultados que producen.

2.4 Alcance

A través del desarrollo de este T.E.G. se pretende mejorar el proceso de OCR contemplado en el proyecto de Documentos Interactivos BCV, explorando el uso de Tecnologías para Computación en Nube, dentro de un escenario de Nube Privada. Las mejoras planteadas están relacionadas con la dimensiones tiempo de procesamiento y precisión en el reconocimiento.

El servicio en nube a desarrollar solo cumplirá con la tarea de procesar los OCR de los documentos cargados mediante el portal web de administración. No se considera la incorporación al servicio otras tareas del sistema, por lo que queda fuera del alcance de este trabajo la evaluación de las mismas.

Ya que se cuenta con la aplicación web de administración en operación y estable, como base para este trabajo, se escapa del alcance las fases de levantamiento de requerimientos, diseño y documentación de la misma, enfocándose únicamente en el diseño y desarrollo del servicio para procesar el OCR.

2.5 Desarrollo de servicio para OCR bajo computación en la nube

El sistema estando en la nube ejecutará las tareas enviadas de manera paralela, valiéndose de las ventajas de la virtualización. La plataforma utilizada en el presente trabajo, CloudCrowd, utiliza un archivo de configuración con formato YAML (YAML Ain't Markup Language, extensión .yml) el cual contiene las credenciales para el protocolo SSH (Secure SHell) y otras parámetros propios del sistema, este archivo brinda la flexibilidad necesaria para que el servicio desarrollado pueda ser implementado sobre otro ambiente de trabajo.

2.5.1 Funcionamiento del servicio en nube

La arquitectura del servicio en nube se muestra en la Figura 14. Donde se pueden observar a nivel macro sus componentes y la forma en que los mismos interactúan con la aplicación administrativa, evidenciando así la integración entre ambas aplicaciones.

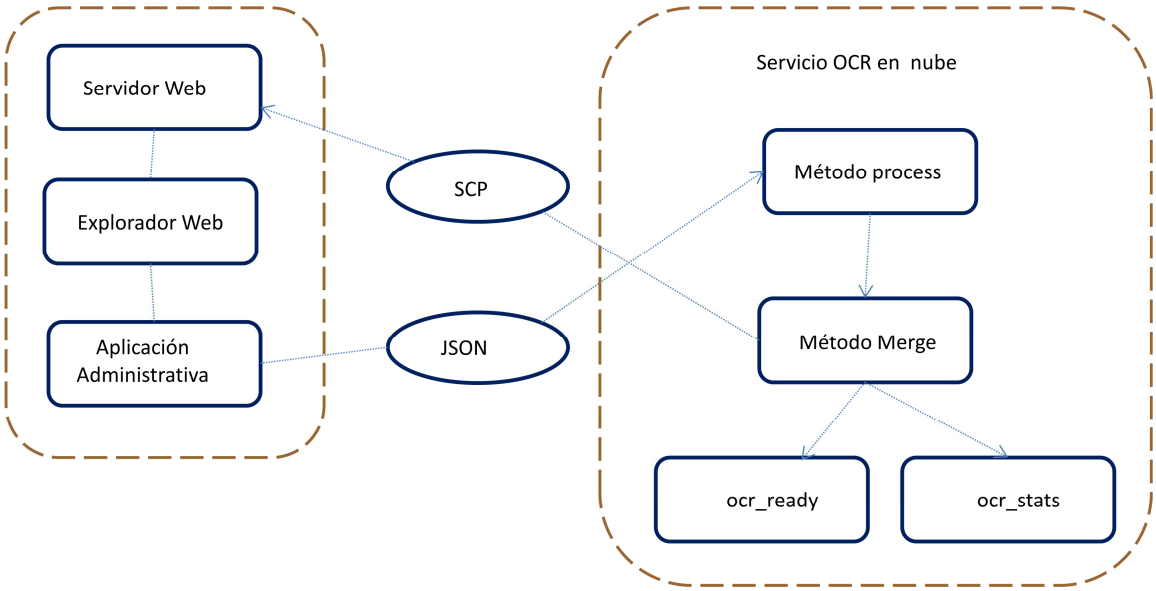

Figura 14*. Arquitectura del Servicio en Nube.*

Los métodos del servicio y su funcionamiento se detallarán a continuación:

Datos de entrada requeridos por el servicio.

Para hacer uso del servicio desarrollado, los documentos deben ser cargados previamente en el servidor de la aplicación web que sirve de cliente a efectos del servicio. En cada petición que se realiza al servicio en la nube se envía un lote de URLs correspondientes a las distintas páginas que posea el documento a procesar.

La estructura de estos URL cumple con las siguientes especificaciones:

- Nombre del documento que se envía al servicio.
- Subdirectorio /jpg dentro del directorio del documento
- Nombre del archivo correspondiente a la página, incluyendo extensión.

Un ejemplo del URL enviado al servicio como entrada es:

BBCV_A100_1941_N2/jpg/BBCV_A100_1941_N2_001.jpg

En un lote enviado al servicio no se mezclan URLs de distintas documentos, por lo que se hacen tantas peticiones como documentos se requiera procesar. Cada llamada que se hace al servicio es vista por el mismo como un nuevo Job. Este Job se encargará luego de iniciar el action o el procedimiento correspondiente al OCR. Es decir, resuelve la petición de la revista asignando páginas de la revista a los distintos nodos de procesamiento de la nube.

Procesamiento de los datos de entrada en el servicio.

Al llegar al servidor de Cloud Crowd se reparten en los distintos nodos cada posición del arreglo de forma automática. A su vez, los nodos toman las imágenes y crean las unidades de trabajo o workers o workers units que son los encargados de llevar a cabo toda la operación de la nube. La acción que ejecutará CloudCrowd por cada Job instanciado consistirá en la ejecución de los métodos split, process y merge. El primero y el último se ejecutan una sola vez por Job, mientras que el segundo ocurre para cada posición del arreglo pasado como entrada al servicio, o bien del arreglo construido y devuelto en el método split. A continuación se explicarán la implementación de los métodos nombrados:

Método split del Action

El action implementado para este propósito no contiene split, ya que el mismo es requerido cuando la entrada a los servicios son argumentos que se deben separar, por ejemplo un archivo comprimido que posea dentro de sí mismo varios archivos. Por tratarse de entradas que contienen la mínima unidad de trabajo (ruta relativa a una imagen) no hace falta lo antes mencionado.

Método process del Action

En la Figura 15 podemos ver de manera gráfica el funcionamiento de este método, el cual será explicado más a fondo.

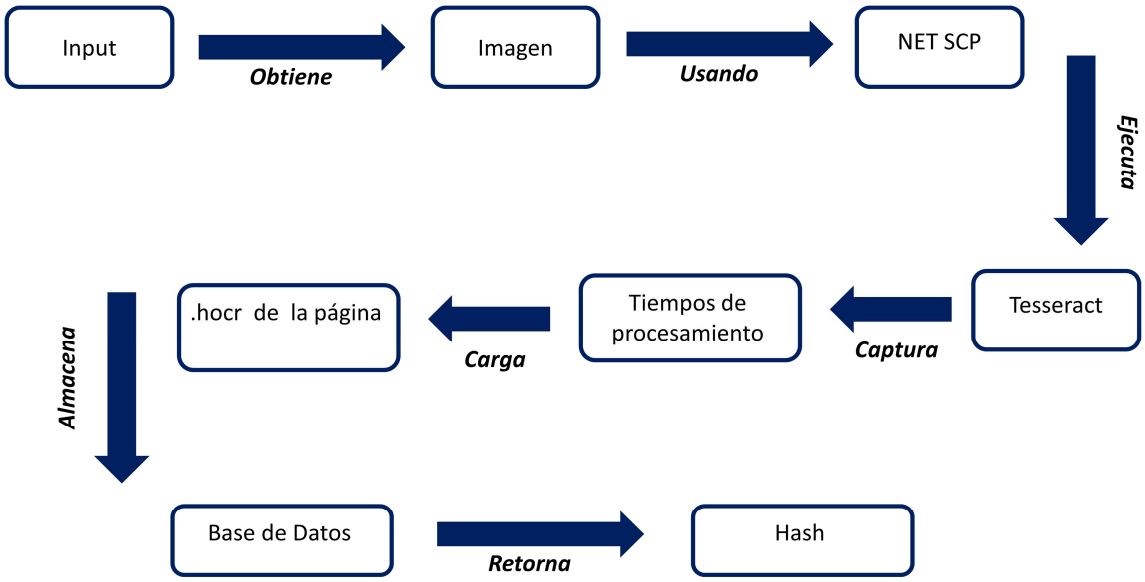

Figura 15. Representación del funcionamiento del método process.

Luego que cada nodo tiene su objeto de entrada definido, se ejecuta el método process definido en el action. Lo primero que hace el nodo es extraer del url el nombre del archivo que contiene la imagen a procesar, además del nombre del documento sobre el cual se está trabajando. En la Figura 16 se puede ver cómo se logra extraer dicha información:

```
def process
  puts input
  arr = input.split(File::SEPARATOR).last(3)
  revista = arr.first
  in_filename = arr.last
```

Figura 16. Extracción de Nombre de revista y Nombre de página.

Todo esto con la finalidad de descargar esta imagen desde el servidor web al cual fue subida, usando para esto un adaptador de Ruby para la utilidad SCP (Secure CoPy), la cual internamente utilizará el protocolo SSH. La manera de emplear dicho adaptador se visualiza en la Figura 17.

```
Net::SCP.start(CloudCrowd.config[:scp_host], CloudCrowd.config[:scp_user], :password => CloudCrowd.config[:scp_password]) do |scp|
  scp.download!(File.join(CloudCrowd.config[:scp_workdir],input), in_filename)
```

Figura 17. Descarga del archivo correspondiente a la imagen a procesar.

En el código anterior se puede observar que se están utilizando para abrir la conexión las credenciales configuradas previamente en el archivo de configuración de Cloud Crowd.

El siguiente paso a ser realizado por el nodo es llamar a Tesseract, el cual es el motor de OCR. Además de eso se toman medidas de tiempo tanto antes de llamar a Tesseract, como después de

completarse su ejecución. Luego de esto, ya se encuentra generada la salida de Tesseract, que consiste de un archivo de extensión .html. En la siguiente figura se detalla dicho proceso:

```
ti = Time.now
ok = system("tesseract #{in_filename} #{in_filename} -l spa -o hocr") # Tesseract appends .html to output filename
tm = Time.now
```

Figura 18. Toma de tiempos y llamada a Tesseract.

El archivo de salida será cargado de vuelta al servidor donde está alojado el documento originalmente, en un subdirectorio predefinido para esto. No se sube el archivo con extensión original sino que, es renombrado para que sea de extensión .hocr y así mantener la integridad con la aplicación existente. Para saber el directorio donde se subirá el archivo se emplea la misma url de entrada al principio de este método en conjunto de un directorio almacenado en el archivo de configuración. Puede apreciarse en la Figura 19 como se hace dicha carga:

```
out_filename = in_filename + ".hocr"
up_dir = File.join(CloudCrowd.config[:scp_workdir],File.split(input)[0],"../ocr",out_filename)

scp.upload!(in_filename + ".html", up_dir)
```

Figura 19. Carga de Resultado de Tesseract.

Una vez subido el archivo, se almacena en base de datos el tiempo de procesamiento del OCR, es decir, el tiempo que tardó el Tesseract en procesar la página. Esta actividad se efectúa una vez por cada página procesada. La base de datos a usar para esto es la configurada en el archivo de configuración de Cloud Crowd, la cual debe ser la misma que la utilizada en la aplicación web al momento de ser incorporado el documento, ya que se asume que las páginas del documento están instanciadas en la tabla correspondiente. La Figura 20 muestra un fragmento de código de cómo se realiza esta tarea:

```
if ok
    p = Pagina.first(:conditions => {
                                      :ruta_de_la_pagina => "public//files//#{revista}//jpg//#{in_filename}"
                                    }
                                  )

    if !p.nil?
      p.seg = tm-ti
      p.save
      puts ". SAVED!"
    else
      puts ". NOT SAVED"
    end
```

Figura 20. Almacenamiento en Base de Datos de tiempo de procesamiento.

Finalmente, se construye la salida del método process, la cual es un objeto de clase Hash. Ese objeto va a contener el nombre del documento, el nombre de la página, un estatus que indicará si fue procesada correctamente, además del tiempo de inicio y fin del procesamiento de la página. Este objeto es serializado antes de ser retornado por el procedimiento, utilizando para esto el formato YAML, nativo en Ruby. Podemos observar lo antes descrito en la Figura 21.

```
      #Objeto output para el Worker que se enviara al Merge
      output[:revista]=revista
      output[:status]=true
      output[:pagina]=in_filename
      output[:ini_time]=ti
      output[:end_time]=tm


    else
      #Objeto output para el Worker que se enviara al Merge
      output[:revista]=revista
      output[:status]=false
      output[:pagina]=in_filename
      output[:ini_time]=ti
      output[:end_time]=tm
    end
    output = YAML::dump(output) #Serializacion con YAML
    return output
```

Figura 21. Retorno del método Process.

Método merge del Action

Ya terminados todos los workers que ejecutaban la tarea process, se activa un nuevo y último worker, el cual se encarga de llevar a cabo lo que esté en el método merge. Dicho método tomará como entrada todos los objetos hash que fueron retornados por los distintos workers que ejecutaron un process. Podemos ver un resumen de este método en la Figura 22:

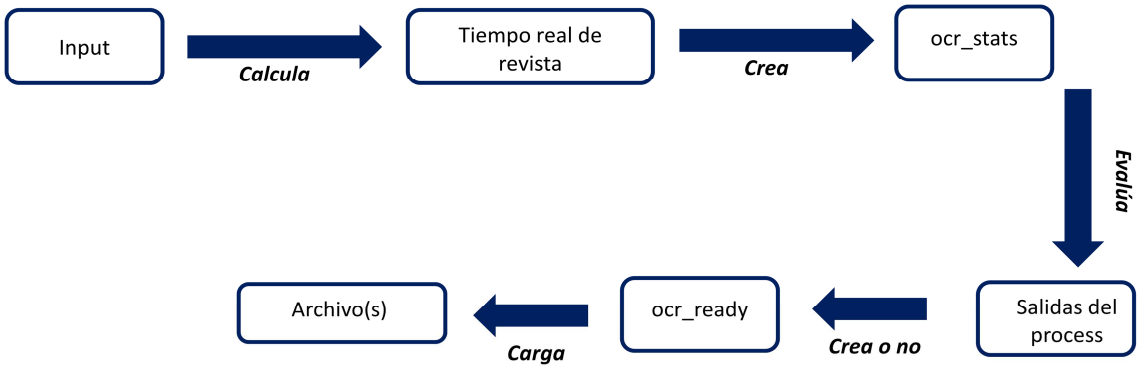

Figura 22. Representación gráfica del método merge.

El merge actúa en sí como un centralizador de resultados, a él llegarán todos los resultados de todos los process que hayan sido ejecutados en la forma de un arreglo. Cada elemento es deserializado con YAML con el fin de obtener el estado de procesamiento de cada página y determinar el tiempo real de procesamiento del documento ya que como anteriormente se mencionó, todos los objetos serializados pertenecen a un mismo documento en cuestión. En código lo anterior se detalla en la Figura 23.

```
input.each do |result|
  result=YAML::load(result) #Deserializacion con YAML
  if i==0
    revista=result[:revista]
    tmp_initial_time=result[:ini_time]
    tmp_final_time=result[:end_time]
  else
    if result[:ini_time] < tmp_initial_time
      tmp_initial_time=result[:ini_time]
    end
    if result[:end_time] > tmp_final_time
      tmp_final_time=result[:end_time]
    end
  end
```

Figura 23. Cálculo de tiempo real de procesamiento de una revista procesada.

Luego esta estadística, sumada a las estadísticas individuales de cada página, es vaciada en un archivo de texto llamado *ocr_stats*. Este archivo es subido al directorio raíz del documento procesado en el servidor donde fue obtenido, usando para esto SCP. La forma como se va armando este archivo ocr_stats se puede observar en la Figura 24:

```
    File.open("ocr_stats","a") do |file|
      file.puts "Pagina: " + result[:pagina] + " --> Status: " + result[:status].to_s + " --> Tiempo Inicial: " + result[:ini_time].to_s
    end


    i+=1

  end

  File.open("ocr_stats","a") do |file|
      file.puts "Tiempo Inicial de Revista --> " + tmp_initial_time.to_s
      file.puts "Tiempo Final de Revista --> " + tmp_final_time.to_s
      file.puts "Tiempo de procesamiento de la revista: " + (tmp_final_time -tmp_initial_time).to_s + " (segundos)"
      file.puts "CANTIDAD DE PAGINAS PROCESADAS --> " + i.to_s
      file.puts "PROMEDIO (Tiempo procesamiento/Paginas) --> " + ((tmp_final_time -tmp_initial_time)/i).to_s + " (segundos)"
  end
```

Figura 24. Creación de Archivo ocr_stats

En la Figura 25 se puede ver un ejemplo de lo que se escribe en el archivo y que será cargado posteriormente:

```
Pagina: BCVOZECO_A15_2010_N1_01_pagina01.JPG --> Status: true --> Tiempo Inicial: Sat May 05 15:06:37 -0430 2012 --> Tiempo Final Sat May 05 15:07:45 -0430 2012
Pagina: BCVOZECO_A15_2010_N1_02_pagina01.JPG --> Status: true --> Tiempo Inicial: Sat May 05 15:07:37 -0430 2012 --> Tiempo Final Sat May 05 15:08:13 -0430 2012
Pagina: BCVOZECO_A15_2010_N1_03_pagina01.JPG --> Status: true --> Tiempo Inicial: Sat May 05 15:06:37 -0430 2012 --> Tiempo Final Sat May 05 15:07:35 -0430 2012
Pagina: BCVOZECO_A15_2010_N1_04_pagina01.JPG --> Status: true --> Tiempo Inicial: Sat May 05 15:07:47 -0430 2012 --> Tiempo Final Sat May 05 15:08:10 -0430 2012
Tiempo Inicial de Revista --> Sat May 05 15:06:37 -0430 2012
Tiempo Final de Revista --> Sat May 05 15:08:13 -0430 2012
Tiempo de procesamiento de la revista: 96.361442 (segundos)
CANTIDAD DE PAGINAS PROCESADAS --> 4
PROMEDIO (Tiempo procesamiento/Paginas) --> 24.0903605 (segundos)
```

Figura 25. Ejemplo de archivo ocr_stats

El worker encargado del merge posteriormente evalúa si todos los objetos que fueron deserializados tuvieron un resultado exitoso, en caso positivo se carga al servidor web un archivo llamado ocr_ready, el cual sirve para indicar que el OCR en esa revista ya fue trabajado, esto se realizó para mantener uniformidad con la aplicación existente. En caso contrario, no subirá este archivo ocr_ready, sino únicamente el archivo anterior ocr_stats. La carga de los archivos se puede evidenciar en la Figura 26:

```
Net::SCP.start(CloudCrowd.config[:scp_host], CloudCrowd.config[:scp_user], :password => CloudCrowd.config[:scp_password]) do |scp|

  up_dir = File.join(CloudCrowd.config[:scp_workdir],revista)
  scp.upload!("ocr_stats",up_dir)

  if flag
    if !File.exist?("ocr_ready")
      File.open("ocr_ready","w")
    end
    scp.upload!("ocr_ready", up_dir)
  end
```

Figura 26. Carga de archivos ocr_stats y ocr_ready.

Descripción del archivo de configuración del servicio.

A continuación se muestra la descripción del archivo de configuración:

```
# This file configures your CloudCrowd installation, and should be consistent
# between your server and all of your nodes. For more information, see:
# http://wiki.github.com/documentcloud/cloud-crowd/the-configuration-folder

:central_server:    http://190.169.69.206:9173
:max_workers:       5
:storage:           filesystem

# Configuracion especifica para RevistasBCV
:scp_host:          190.169.69.206
:scp_user:          tesistas
:scp_password:      tesistas
:scp_workdir:       /home/tesistas/bcvadmin/public/files
:activerecord_adapter:  mysql
:activerecord_db:       bcv_revs_crowd
:activerecord_host:     190.169.69.206
:activerecord_login:    root
:activerecord_password: ccpdroot
```

Figura 27. Ejemplo de archivo de configuración para el servicio.

Cliente del servicio de reconocimiento óptico de caracteres en la nube

Para poder utilizar el servicio desarrollado se empleó la aplicación web de administración existente anteriormente en el proyecto. Se incorporó al menú principal una opción que permitiese llamar al servicio OCR en la nube y para esto se añadió en la base de datos el módulo OCR. El menú resultante de esta actualización se puede apreciar en la Figura 28, donde se muestra en el extremo inferior izquierdo la opción OCR incorporada.

Figura 28. Opción para realizar el OCR en la aplicación web.

Para agregar la opción OCR dentro de la aplicación actual, se contó con la elaboración de un controlador y de su respectiva vista, respetando así el patrón propio de Rails MVC (Modelo-Vista-Controlador por sus siglas).

Vista

Para el desarrollo de la vista se consideró el modelo de casos de uso de la Figura 29.

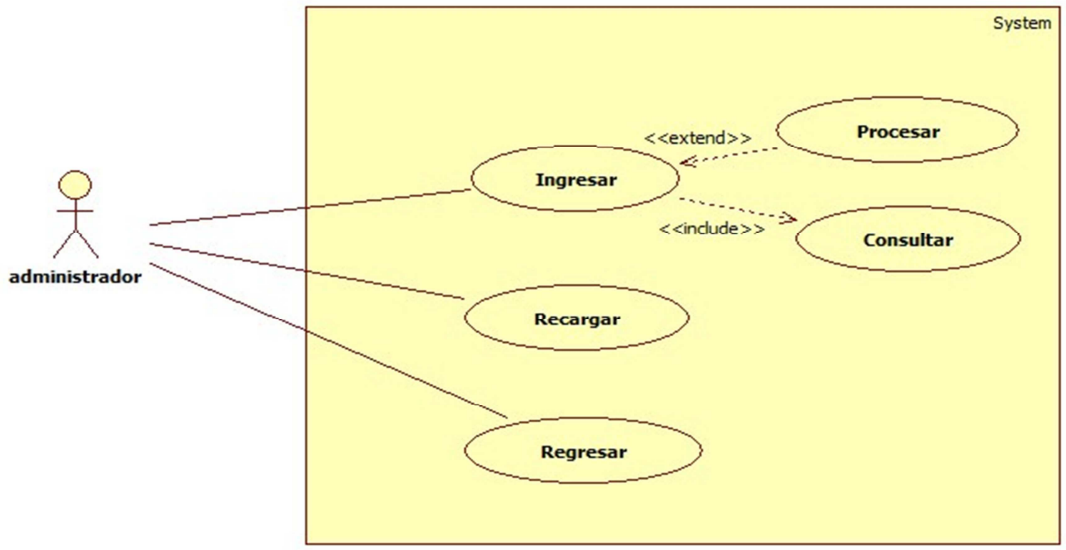

Figura 29*. Modelo de Casos de Uso*

La aplicación anterior tomaba la totalidad de los documentos cuyo proceso OCR no estuviera realizado y los mandaba a procesar, sin dejar la opción al usuario de seleccionarlos. Este comportamiento se modificó en el cliente del servicio, permitiendo al usuario seleccionar cuáles documentos desea procesar entre todos los disponibles.

Adicionalmente se debe mostrar al usuario el progreso que tienen los documentos que ya han sido enviados al servidor. Cabe destacar, que este progreso será consultado cada vez que se ingrese al módulo OCR, o desde la misma pantalla donde se listan los documentos con su funcionalidad de *Recargar.* Cuando un documento es consultado por su progreso y ya ha sido completado por el servicio en la nube, no podrá ser enviado al servicio.

En la Figura 30 podemos observar la manera que quedó plasmada en la interfaz desarrollada, estas funcionalidades.

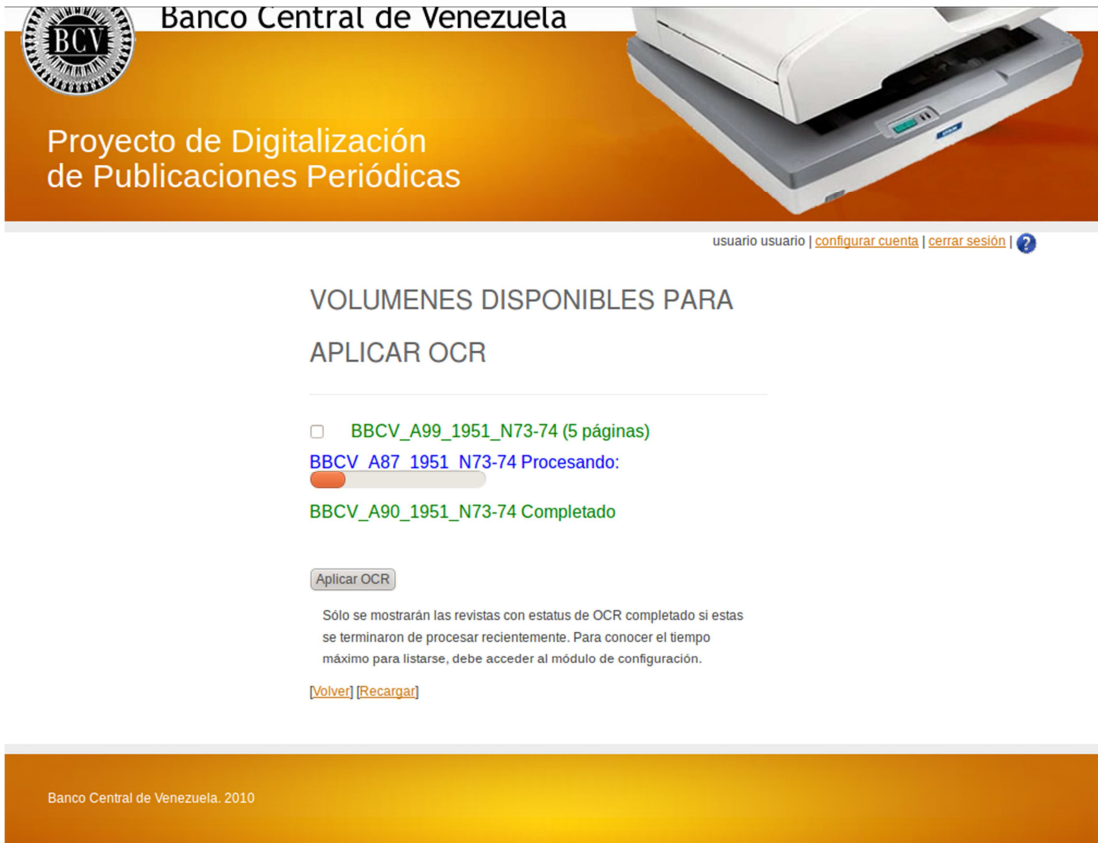

Figura 30. ***Vista de la Aplicación.***

Controlador

El controlador desarrollado cuenta con tres (3) funciones:

- Listar los documentos disponibles para ser enviados al servicio
- Listar y determinar el progreso de los documentos que hayan sido enviados al servicio de OCR en nube.
- Realizar la llamada al servicio en nube para procesar los documentos.

Lista de documentos que se desplegarán en la vista.

Para determinar cuáles documentos serán enviados a la vista del módulo en el controlador se cuenta con dos (2) listas que identifican cada caso a ser tratado. En caso de que un documento haya sido terminado de procesar por el servicio, se mostrará como completado, mas sin embargo, sólo aquellos documentos que hayan sido terminados en un tiempo menor a *max_seg_mostrar* serán los listados para su posterior uso en la vista del módulo. El tiempo con el que se evalúa es un

parámetro configurable dentro de la aplicación. Aunque un documento no sea mostrado por la expiración del tiempo, no quiere decir que se haya eliminado la información de procesamiento, ni que el documento haya sido desincorporado, el mismo puede ser localizado desde el módulo de Volúmenes, como se realiza actualmente.

Para determinar cuáles documentos asignar a cada lista, el controlador evalúa la existencia de los archivos ocr_ready y ocr_progress. En caso de no existir ninguno de los dos (2) archivos se listará como un documento sin OCR, y en caso de sólo poseer el archivo ocr_progress se listará como un documento que está siendo procesado actualmente por el servicio en la nube. Esta evaluación se puede observar en el siguiente segmento de código:

```
if !File.exist?("#{@@ruta}/#{revista}/ocr_ready") and !File.exist?("#{@@ruta}/#{revista}/ocr_progress")
        @revistas_sin_ocr.push revista
      elsif File.exist?("#{@@ruta}/#{revista}/ocr_progress")
.
.
.
.
@revistas_en_progreso[revista] = progress
```

Figura 31. Evaluación de tipo de documento dentro del controlador.

Procesamiento de documentos

Llamada al servicio en nube para procesar el OCR de los documentos seleccionados.

Los documentos previamente seleccionados por el usuario en la vista y que se deseen enviar al servicio en nube son obtenidos en el método procesar_ocr dentro del controlador, tal como se observa en la Figura 32.

```
def procesar_ocr
  rs = params[:revistas]
```

Figura 32. Captura de documentos a enviar al servicio en nube.

Posteriormente se listan todas las rutas relativas correspondientes a todas las páginas de dicho documento y se almacenan en un arreglo. Seguidamente, se crea el subdirectorio OCR dentro del directorio del documento seleccionado. En este subdirectorio serán almacenados los resultados del OCR por parte del servicio en la nube. Todo esto lo observamos en el siguiente código:

```
rs.each_value do |r|
  @in_imgs = Array.new()

  imagenes = Dir.open(File.join(@@ruta,r,"jpg"))

  imagenes.sort.each do |img|

    if !@@ignore_list.include? img then
      nombre=File.join(r,"jpg",File.basename(img))
      @in_imgs.push nombre
    end

  end



  FileUtils.mkdir("#{@@ruta}/#{r}/ocr") if !File.exist?("#{@@ruta}/#{r}/ocr")
```

Figura 33. Lote de URLs que se enviará al servicio y creación del directorio /ocr.

Una vez se haya creado el directorio se procede a invocar al servicio en la nube, utilizando para ello dos (2) gemas, RestClient para enviar peticiones al servidor y JSON (JavaScript Object Notation) para que los datos enviados viajen con este formato de datos. La llamada en detalle se puede ver a continuación:

```
request = RestClient.post('http://localhost:9173/jobs',
    {:job => {
        'action' => 'tesseract_ocr',
        'inputs' => @in_imgs,
        'options' => {
            'batch_size' => 1
        }
    }.to_json}
)
```

Figura 34. Llamada al servicio en la nube desde el controlador.

La llamada antes descrita se compone de la siguiente manera:

- Dirección del servidor que presta la funcionalidad de OCR en la nube
- Objeto Hash llamado job, que debe poseer los atributos:
    - Action: Nombre del Action a ejecutar en la nube.
    - Inputs: Arreglo con las URLs de las páginas que son enviadas al servicio
    - Options: Distintas opciones propias del servicio
- El Objeto luego de ser descrito, debe ser transformado a un objeto JSON.

Seguidamente que se llama al servicio se crea un archivo de texto llamado ocr_progress el cual es usado para monitorear el avance de procesamiento de la revista.

Al usuario por último se le muestran de manera informativa las tareas que él mandó a procesar con anterioridad.

Estructura del archivo ocr_progress.

La estructura del archivo anteriormente creado por el controlador cumple con las siguientes especificaciones:

- ID del job obtenido al momento de la llamada al servicio.
- Estado de procesamiento actual para el documento en el servicio.
- Porcentaje de avance en el procesamiento del documento.
- Tiempo transcurrido desde que inicio la tarea.
- Salidas de la tarea por parte del servicio

En la Figura 35 se puede apreciar un ejemplo del contenido de este archivo:

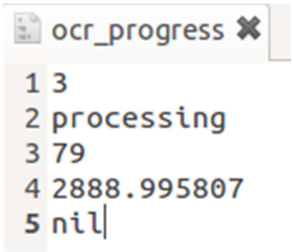

Figura 35. Contenido de archivo ocr_progress

Progreso de documentos enviados al servicio.

En el caso de que el documento haya sido procesado y el usuario ingrese al módulo OCR se llama al servicio en la nube con el id del job correspondiente al documento el cual se obtiene de la lectura del archivo ocr_progress, como podemos observar en el código de la Figura 36 a continuación:

```
File.open("#{@@ruta}/#{revista}/ocr_progress","r") do |file|
  id = file.gets.to_i
  status = file.gets.strip
  progress_old = file.gets.to_i
end

if status != "failed"
  progress = job_progress(id)
end
```

Figura 36. Llamada al servicio para obtener el progreso de un documento.

En caso de que ocurra un error en esta consulta se le muestra un mensaje de error al usuario que será mostrado en el apartado de la vista.

Método job_progress

Este método se encarga de solicitar al servidor de la nube la información correspondiente a un job que se encuentre en proceso actualmente. Para obtener dicha información se emplea RestClient y su método get, el cual va y obtiene el objeto representativo de una URL específica, dicho objeto es capturado en formato JSON por lo que debe posteriormente ser convertido para que se pueda trabajar con él. El código de este método se muestra en la siguiente figura:

```
def job_progress(id)

  url = "http://localhost:9173/jobs/"+id.to_s

  result = {}

  begin
    job_representation = RestClient.get url
    puts "Resource con el get: "
    puts job_representation
    job_representation = JSON.parse(job_representation)
    result[:valid] = true
    result[:status] = job_representation['status']
    result[:percent] = job_representation['percent_complete']
    result[:uptime] = job_representation['time_taken']
    result[:outputs] = job_representation['outputs']
  rescue => e

    result[:valid] = false
  end
```

Figura 37. Código de método job_progress

2.6 Experimentos y resultados

Para evaluar las mejoras obtenidas se diseñaron pruebas que permitiesen comparar el módulo OCR existente en el sistema actual con el Servicio en nube desarrollado en este T.E.G. Para la comparación de ambas versiones se evaluaron dos variables fundamentales:

- Tiempo de procesamiento por documento (TPD)

- Precisión de reconocimiento del proceso OCR (POCR).

Para realizar las pruebas se utilizó la misma plataforma de hardware sobre la que se realizó la carga inicial de documentos en el sistema actual. La misma consiste de un servidor ubicado en el Centro de Computación Paralela y Distribuida de la Facultad de Ciencias de la UCV, con las siguientes características:

- Marca: HP
- Modelo: ProLiant ML150
- CPU: Intel® Xeon® Processor E5405 @ 2.0Ghz Quad Core
- Memoria: 2x6MB L2 Cache; 4GB DDR2-667 RAM

Con respecto a la implementación de la nube utilizada para estas pruebas, se emplearon tres máquinas virtuales dentro del servidor de trabajo, cada una de ellas cumplía las funciones de un nodo dentro de la nube. Para dicha virtualización se utilizó la herramienta Virtual Box de Oracle, colocando como sistema operativo Ubuntu 10.04 "Lucid Lynx" en cada una de las máquinas virtuales, así como la suite de herramientas necesarias para poder iniciar un nodo de la nube. Para iniciar cada instancia de máquina virtual a usarse en las pruebas, se usó el siguiente comando, el cual nos omitió el uso de la interfaz gráfica con el fin de reducir cualquier carga extra al sistema:

VBoxHeadless -startvm <nombre_de_la_máquina_virtual> &

2.6.1 Pruebas para medición de tiempo de procesamiento por documento

Esta prueba consiste en comparar el tiempo tomado para realizar el OCR en los documentos por el sistema actual al momento de realizar la carga inicial de los mismos en la Base de Datos, con el tiempo que toma el servicio en nube para realzar el OCR de los mismos documentos.

Muestra para medir tiempo de procesamiento por documento

El conjunto de prueba consiste de un lote de diez documentos pertenecientes a diferentes décadas seleccionados entre los que se encontraban cargados en el sistema actual, de manera que la muestra fuese heterogénea con respecto a la fecha de publicación de los documentos. La distribución de documentos por décadas se muestra en la Figura 38:

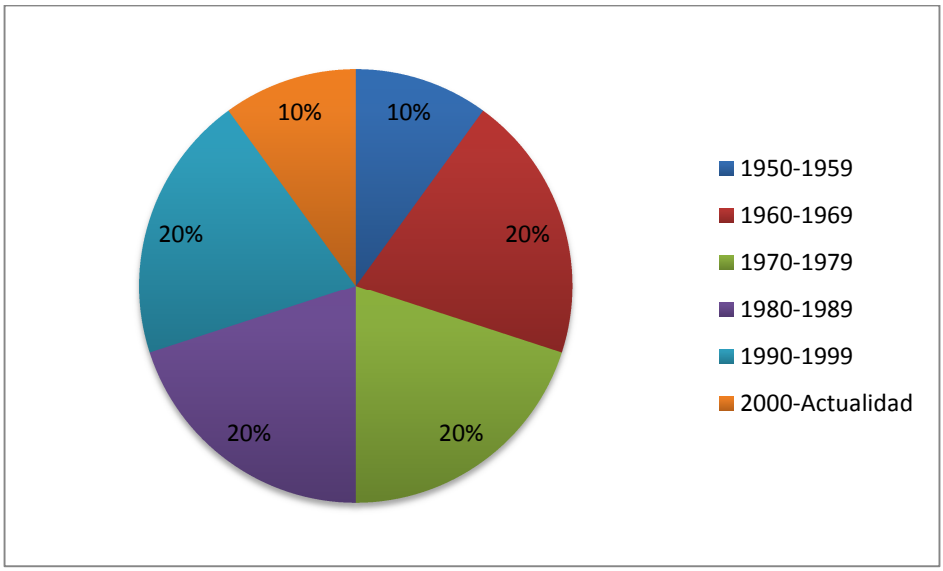

Figura 38. Distribución por décadas de documentos seleccionados para pruebas.

Entre estos diez documentos se procesaron en total dos mil trece páginas en el servicio en la nube. La distribución de estas páginas en los documentos tomados como muestra se puede observar en la tabla de la Figura 39.

Nombre del Documento	N° de Páginas	Década	Páginas por década	% Páginas por década
BBCV_A14_1954_N107-109	112	1950-1959	112	5,56%
REVISTABCV_A26_1966_N260-262	273	1960-1969	413	20,52%
REVISTABCV_A23_1963_N224-226	140			
RELA_A10_1974_N39	208	1970-1979	452	22,45%
RELA_A11_1975_N42	244			
RBCENTRALV_A1_1986_N4	290	1980-1989	465	23,10%
RBCENTRALV_A1_1986_N3	175			
RBCV_A6_1991_N2	342	1990-1999	567	28,17%
RBCV_A5_1990_N2	225			
BCVOZECO_A15_2010_N1	4	2000-Actualidad	4	0,20%
Total	2013		2013	100%

Figura 39. Tabla de distribución por década de páginas enviadas al servicio en nube.

Ejecución de las pruebas de tiempo de procesamiento OCR por documento

Para las pruebas de tiempo de procesamiento por documento se utilizó como referencia los valores existentes en la Base de Datos para las páginas procesadas con el sistema actual. Se inició

el procesamiento controlado de los documentos seleccionados en la nube, para así generar la información de tiempo en el archivo ocr_stats, creado para tales propósitos.

La información de tiempo de procesamiento almacenado en la Base de Datos fue extraída aplicando la siguiente sentencia de SQL:

SELECT p.revista_id,SUM(p.`seg`) FROM bcv_revistas.paginas p WHERE p.revista_id IN (<id_revista>[,<id_revista>]) GROUP BY p.revista_id;

Esto nos permitió obtener la suma de los tiempos de procesamiento de las página por cada documento en el sistema actual.

Para la obtención de los tiempos de procesamiento para el servicio en la nube se realizó el procesamiento cada uno de los documentos por separado, con el fin de tener una igualdad de condiciones en las pruebas. Esto se logró utilizando el módulo de OCR creado en el portal web de la aplicación, enviando un documento a la nube y esperando a su finalización para continuar con el siguiente documento.

Una vez ejecutadas las pruebas desde la interfaz web, se continuó recolectando los archivos ocr_stats generados por la nube al terminar el procesamiento. La ubicación de dicho archivo está cubierta en la sección 2.5.1. De dichos archivos se obtuvo el tiempo de procesamiento total del documento, tomando para esto el tiempo ocurrido desde que se inició la primera página hasta que se terminó la última página pendiente por procesar, indistintamente del nodo en que se hayan procesado las páginas.

Resultados obtenidos.

Como unidad de medida en esta parte de las pruebas, se utilizó la cantidad de segundos transcurridos desde el instante en que se inicia en la nube el primer procesamiento de OCR, hasta el momento en que culmina la última tarea de OCR pendiente, indistintamente del nodo o el worker que haya realizado tales tareas.

También se incluye un valor derivado de estas mediciones bajo el nombre de Porcentaje de Aceleración, el cual está determinado por el porcentaje de tiempo ahorrado en la aplicación como servicio en nube, basado en el tiempo de procesamiento consumido en el sistema actual.

En la tabla de la Figura 40 se muestra la comparación del tiempo tomado por el sistema actual, así como el resultado obtenido por su homólogo procesado en la nube. En la Figura 41 se muestra esta comparación en gráfico de barras.

Nombre del Documento	Tiempo (segundos)		% Aceleración
	Actual	Servicio en Nube	
BBCV_A14_1954_N107-109	4862	2830	41,79%
REVISTABCV_A26_1966_N260-262	5578	3390	39,23%
REVISTABCV_A23_1963_N224-226	2530	2001	20,91%
RELA_A10_1974_N39	1395	977	29,96%
RELA_A11_1975_N42	2269	1279	43,63%
RBCENTRALV_A1_1986_N4	1686	1452	13,88%
RBCENTRALV_A1_1986_N3	793	778	1,89%
RBCV_A6_1991_N2	2714	2397	11,68%
RBCV_A5_1990_N2	1448	1102	23,90%
BCVOZECO_A15_2010_N1	69	96	-39,13%
Total	23344	16302	18,77%

Figura 40. Resultados de ambas versiones probadas.

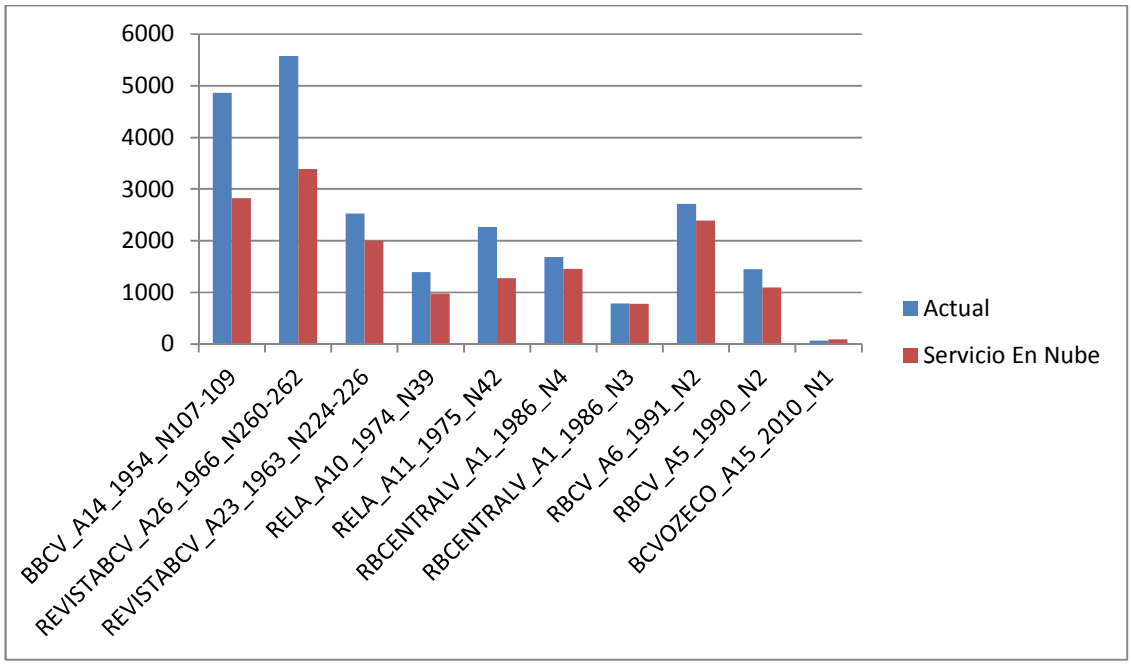

Figura 41. Tiempo de procesamiento de ambas versiones basado en segundos.

En la figura anterior se muestra que los tiempos de procesamiento obtenidos fueron en general inferiores en versión que utiliza el servicio en la nube. El mayor porcentaje de aceleración fue de 43,63% en el documento RELA_A11_1975_N42, mientras que el porcentaje más bajo fue de -39,13%, en el caso del documento BCVOZECO_A15_2010_N1. De diez documentos procesados,

nueve tuvieron una mejoría positiva, mientras que solo uno tuvo un peor desempeño en la versión del servicio en la nube. En este último caso se debe considerar que la cantidad de páginas del documento no era grande, al punto de que no llegó a ocupar todos los workers disponibles en la nube.

Evaluación en distintas configuraciones de la nube

Para estas pruebas se tomaron cuatro documentos del conjunto de diez evaluado anteriormente, y se realizaron mediciones para comparar los tiempos que se obtuvieron para ellos con tres configuraciones distintas.

Documento	Tres Nodos y Cinco Workers		
	Tiempo (Segundos)		% Aceleración
	Actual	En Nube	
BBCV_A14_1954_N107-109	4862	2830	41,79%
REVISTABCV_A26_1966_N260-262	5578	3390	39,23%
RBCENTRALV_A1_1986_N3	793	778	1,89%
BCVOZECO_A15_2010_N1	69	96	-39,13%

Figura 42. Tiempos de Procesamiento utilizando tres nodos y cinco workers.

Documento	Tres Nodos y Diez Workers		
	Tiempo (en segundos)		% Aceleración
	Actual	En Nube	
BBCV_A14_1954_N107-109	4862	2872	40,93%
REVISTABCV_A26_1966_N260-262	5578	3426	38,58%
RBCENTRALV_A1_1986_N3	793	775	2,27%
BCVOZECO_A15_2010_N1	69	95	-37,68%

Figura 43. Tiempos de Procesamiento utilizando tres nodos y diez workers

Documento	Dos Nodos y Cinco Workers		
	Tiempo (en segundos)		% Aceleración
	Actual	En Nube	
BBCV_A14_1954_N107-109	4862	3811	21,62%
REVISTABCV_A26_1966_N260-262	5578	4777	14,36%
RBCENTRALV_A1_1986_N3	793	1033	-30,26%
BCVOZECO_A15_2010_N1	69	95	-37,68%

Figura 44. Tiempos de Procesamiento utilizando dos nodos y cinco workers

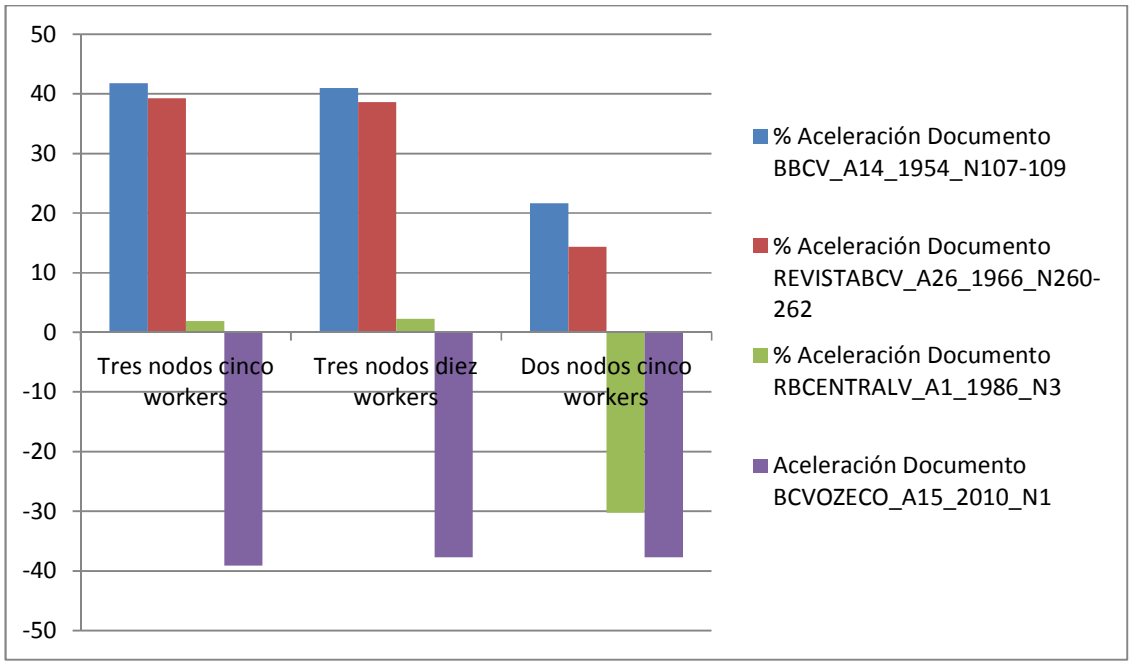

Figura 45. Comparativa de porcentajes de Aceleración en configuraciones de nube distintas

En la Figura 42, Figura 43 y Figura 44, además del gráfico en la Figura 45 se puede apreciar que las configuraciones con tres nodos tuvieron mejores resultados en lo referente a la aceleración del tiempo de procesamiento OCR, mientras que la configuración de dos nodos no tuvo un porcentaje de aceleración que superase a estos dos anteriores. Por esta razón se decide dejar como configuración del sistema en nube la de tres nodos y cinco workers.

2.6.2 Pruebas para medición de precisión del proceso OCR

Para estas pruebas se transcribió muy cuidadosamente los textos de un total de trece páginas seleccionadas del subconjunto de siete documentos entre los diez documentos enviados a procesar en el servicio, que tenían el texto OCR cargado en la Base de Datos de páginas del sistema actual. Estos textos transcritos y verificados fueron utilizados como referencia, o textos correctos, para realizar la evaluación de precisión de reconocimiento, tanto para el motor OCR utilizado en el sistema actual como para el servicio en la nube implementado.

Las páginas transcritas quedaron distribuidas por la década a la que pertenecen como se muestra en la Figura 46.

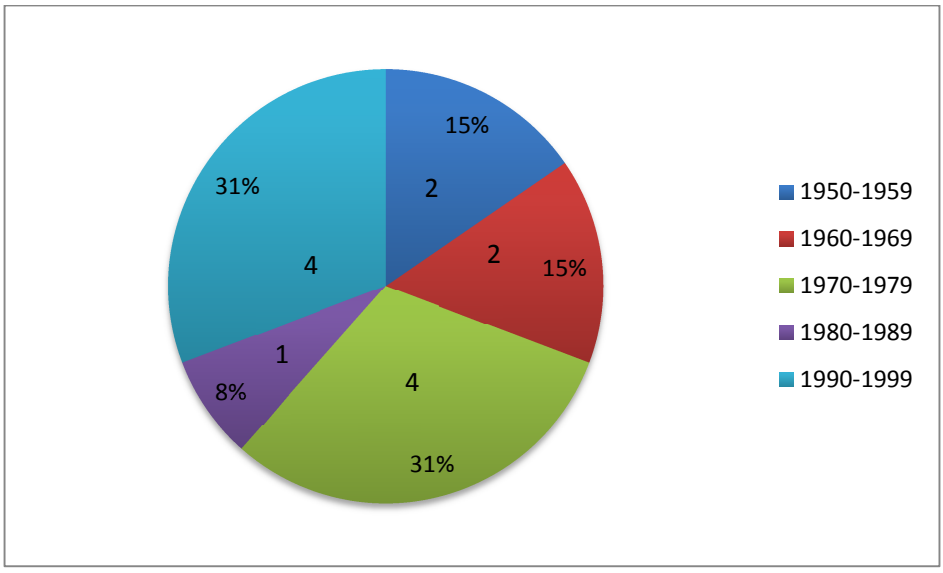

Figura 46. Distribución por décadas de las páginas transcritas para pruebas.

Para medir la precisión del motor OCR se empleó una herramienta específica para esta tarea, desarrollada por la Universidad De Nevada, Los Ángeles (UNLV por sus siglas en inglés), encargada de llevar a cabo las pruebas para motores OCR.

La prueba para la precisión del motor OCR se dividió en dos fases:

- Precisión de reconocimiento de caracteres, donde se compararon ambas versiones y su capacidad para reconocer correctamente los patrones de caracteres dentro de una imagen procesada.
- Precisión de reconocimiento de palabras, donde el objetivo era comparar la potencialidad a la hora de identificar palabras dentro de la imagen de forma correcta.

Para llevar a cabo las pruebas se debió extraer el texto del archivo con resultados OCR (.hocr), tanto el existente como el nuevo generado por el servicio en la nube, mediante el empleo de la herramienta hocr2pdf. El comando que se empleó para estos fines tiene la siguiente estructura:

```
hocr2pdf -i <Nombre_pagina>.jpg -o <Nombre_pagina>.pdf < <Nombre_pagina>.jpg.hocr -t <Nombre_pagina>.txt
```

Una vez obtenidos los archivos de texto correspondientes al contenido de los archivos .hocr de cada modelo, se hizo uso de los programas *wordacc* y *accuracy* que forman parte del sistema de la UNLV.

La manera en que fueron empleados dichos programas fue:

1- Medición de Precisión de reconocimiento de caracteres

```
accuracy <Nombre_pagina_correcta>.txt <Nombre_pagina_ocr>.txt report_char_<cloud|old>.txt
```

2- Medición de Precisión de reconocimiento de palabras

```
wordacc <Nombre_pagina_correcta>.txt <Nombre_pagina_ocr>.txt report_word_<cloud|old>.txt
```

Resultados correspondientes a la precisión de reconocimiento de caracteres

Para la evaluación de los resultados se utilizaron los siguientes criterios:

- Caracteres correctos: Caracteres dentro del texto transcrito de la página, que están presentes dentro del texto OCR del modelo y cuyo orden de aparición concuerda con el transcrito.
- Caracteres incorrectos: Cantidad de operaciones de eliminación, substitución o agregación de caracteres que deba hacerse al texto OCR para que sea igual al texto transcrito.
- Porcentaje de Precisión:
- Porcentaje de Error:

En la Figura 47 se observan los resultados obtenidos por precisión de reconocimiento de caracteres por cada página y en la Figura 48 los resultados generales.

Página	**Sistema Actual**				**Servicio en la Nube**			
	Correctos	**Errores**	**%Precisión**	**%Error**	**Correctos**	**Errores**	**%Precisión**	**%Error**
1	1125	485	69,88	30,12	1502	108	93,29	6,71
2	4302	502	89,55	10,45	4490	314	93,46	6,54
3	289	1356	17,57	82,43	543	1102	33,01	66,99
4	943	467	66,88	33,12	1379	31	97,80	2,2
5	2457	169	93,56	6,44	2603	23	99,12	0,88
6	94	2533	3,58	96,42	2454	173	93,41	6,59
7	1311	586	69,11	30,89	1755	142	92,51	7,49
8	2260	158	93,47	6,53	2305	113	95,33	4,67
9	953	287	76,85	23,15	1154	86	93,06	6,94
10	1454	59	96,10	3,9	1454	59	96,10	3,9
11	275	527	34,29	65,71	727	75	90,65	9,35

12	3055	835	78,53	21,47	3539	351	90,98	9,02
13	968	292	76,83	23,17	1232	28	97,78	2,22

Figura 47. Resultados de precisión de reconocimiento de caracteres por cada página

Versión	Correctos	Errores	%Precisión	%Error
Sistema Actual	19.486	8.256	70,24	29,76
Servicio en Nube	25.137	2.605	90,61	9,39

Figura 48. Resultados generales para todas las páginas

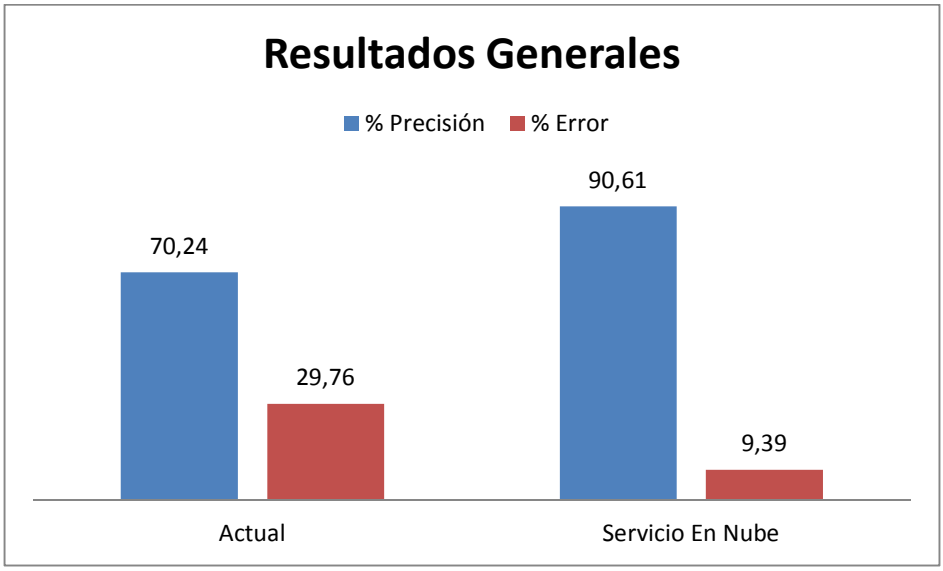

Figura 49. Comportamiento de los modelos con respecto al porcentaje de caracteres correctos e incorrectos.

Como se puede observar en la Figura 49 y según los resultados obtenidos luego de las pruebas el modelo que mejor precisión posee es el servicio en la nube que se desarrolló, arrojando un porcentaje de precisión del 90,61% y un porcentaje de error del 9,39%, lo que representa un incremento del 29% en precisión y una disminución del 68,45% en errores respecto al sistema actual.

Resultados correspondientes a la precisión de reconocimiento de palabras

Para la evaluación de los resultados se utilizaron los siguientes criterios:

- Palabras correctas: Palabras dentro del texto transcrito de la página, que están presentes dentro del texto OCR del modelo y cuyo orden de aparición concuerda con el transcrito.
- Palabras incorrectas: Palabras dentro del texto OCR que no son mostrados dentro del texto transcrito, o que no respetan la secuencia de aparición correcta.
- Porcentaje de Precisión: $\mathbf{100} - \boldsymbol{Porcentaje\ de\ Error}$

- Porcentaje de Error:

  $$cantidad\ de\ palabras\ incorrectas * 100\ /$$
  $$Total\ de\ Palabras\ en\ Texto\ Transcrito$$

En las siguientes tablas, se observarán los resultados obtenidos por precisión de reconocimiento de palabras por cada página y resultados generales.

Página	Sistema Actual				Servicio en la Nube			
	Correctos	Errores	%Precisión	%Error	Correctos	Errores	%Precisión	%Error
1	153	76	66,81	33,19	213	16	93,01	6,99
2	537	205	72,37	27,63	594	148	80,05	19,95
3	18	32	36	64	46	4	92	8
4	133	67	66,5	33,5	187	13	93,5	6,5
5	330	112	74,66	25,34	426	16	96,38	3,62
6	78	326	19,31	80,69	319	85	78,96	21,04
7	172	85	66,93	33,07	205	52	79,77	20,23
8	286	83	77,51	22,49	307	62	83,20	16,8
9	114	56	67,06	32,94	139	31	81,76	18,24
10	198	35	84,98	15,02	204	29	87,55	12,45
11	52	23	69,33	30,67	72	3	96	4
12	397	205	65,95	34,05	463	139	76,91	23,09
13	142	56	71,72	28,28	190	8	95,96	4,04

Figura 50. Resultados de precisión de reconocimiento de caracteres por cada página

Versión	Correctos	Errores	%Precisión	%Error
Sistema Actual	2.610	1.361	65,73	34,27
Servicio En Nube	3.365	606	84,74	15,26

Figura 51. Resultados de precisión de reconocimiento de caracteres por cada página

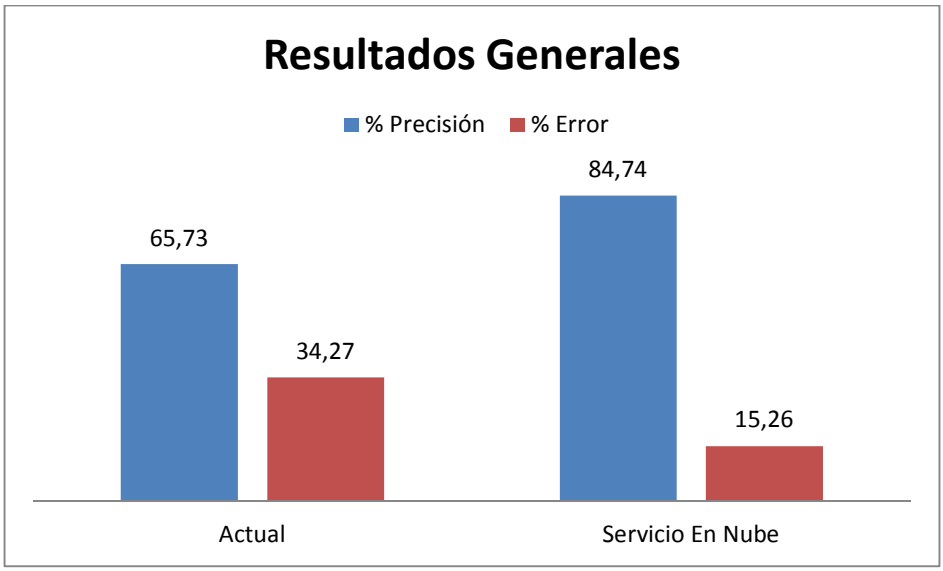

Figura 52. Comportamiento de los modelos con respecto al porcentaje de palabras correctas e incorrectas.

Como se puede observar en la Figura 52, los resultados obtenidos luego de las pruebas sobre el reconocimiento de palabras del motor OCR, el modelo que mejor precisión posee es el servicio en la nube, arrojando un porcentaje de precisión del 84,74% y un porcentaje de error del 15,26%, lo que representa un incremento del 28,92% en precisión y una disminución del 55,47% en errores respecto al modelo existente.

CONCLUSIONES

En este Trabajo Especial de Grado se desarrolló una aplicación OCR bajo el enfoque de computación en la nube, la cual se integró a la aplicación web de administración de la versión del proyecto de Documentos Interactivos BCV que se encuentra instalada en el Centro de Computación Paralela y Distribuida de la Universidad Central de Venezuela, empleando la misma plataforma.

Las pruebas realizadas revelaron una disminución del tiempo de procesamiento en el servicio en la nube respecto al sistema actual. La alternativa empleada justifica su utilización para grandes volúmenes de datos, ya que explota las potencialidades del hardware dedicado al procesamiento OCR.

También se evidenció una diferencia en la precisión de los motores OCR empleados en ambas versiones, siendo mejores los resultados obtenidos en el servicio en la nube que emplea una versión más actualizada del motor, dejando muestra de los avances que se tienen en un corto tiempo en el área del reconocimiento de caracteres, que sigue siendo un campo de investigación en la ciencia de la computación.

Aun cuando sólo se trabajó sobre una tarea de las que conforma el sistema actual, estas mejoras pueden impactar en un mejor desempeño del sistema en general, ya que otras funcionalidades realizadas por el sistema se verían beneficiadas con la nueva aplicación, por ejemplo búsquedas sobre los documentos por palabras claves resultarían ahora más precisas. Todo esto dado que la disminución del tiempo que se logró en el proceso OCR permite la indexación de nuevos documentos y el reprocesamiento OCR de los ya indexados en tiempos aceptables.

El trabajo aquí realizado deja abierta la posibilidad de añadir nuevas funcionalidades a la nube que no se realizan actualmente bajo este enfoque, queda de futuros trabajos lograr integrar con el servicio desarrollado las otras tareas, obteniendo un sistema completamente en nube y así explotar las capacidades de la computación en la nube en la totalidad del sistema. El uso del patrón Modelo-Vista-Controlador apoyará este proceso de transición entre el sistema actual y los futuros componentes en la nube.

Las aplicaciones en nube privada como este caso, son un camino viable para todas las organizaciones que desean mejorar sus sistemas actuales. Es un tipo de nube que permite tener el control interno de los datos y por consiguiente un mayor control sobre lo que se desarrolla. La

computación en la nube debe ser un enfoque que en los próximos años sea un pilar para las tecnologías de la información y debe ser considerado siempre que se quieran tener sistemas eficientes en el uso de los recursos computacionales.

BIBLIOGRAFÍA

- [1] H. Bunke, P.S.P. Wang. **Handbook of Character Recognition and Document Image Analysis.** World Scientific Publishing Co. Pte. Ltd. Singapore (1997).
- [2] M. Cheriet, N. Kharma, C. Liu, C. Suen. **Character Recognition Systems: A Guide for Students and Practitioners.** Wiley Interscience. New Jersey (1999).
- [5] M. Ambrust, et al., "**Above the Clouds: A Berkeley View of Cloud Computing**" Electrical Engineering and Computer Sciences, Univers ity of California at Berkeley, California,Technical Report UCB/EECS-2009-28, 2009.
- [6] S. Bennett, M. Bhuller, and R. Covington, "**Architectural Strategies for Cloud Computing**" Oracle Corporation, 2009.
- [7] S. Kajeepeta, "**Cloud Computing: From Metaphor to Mainstream,**" Software Magazine, vol. 27, no. 6, pp. 10-13, Nov. 2008.
- [8] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, "**Cloud Computing**," IBM Corporation, 2007.
- [9] D. Quan, "From Cloud Computing to the New Enterprise Data Center," IBM Corporation, 2008.
- [10] J. D. Lasica, **Identity in the Age of Cloud Computing**. United States of America: The Aspen Institute, 2009.
- [11] R. Mikkilineni and V. Sarathy, "**Cloud Computing and the Lessons from the Past**," in 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009, pp. 57-62.
- [12] B. Ohlman, A. Eriksson, and R. Rembarz, "**What Networking of Information Can Do for Cloud Computing,**" in 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009, pp. 78-83.
- [13] A. Kamaraju and P. Nicolas, "**Cloud Storage**," Storage Networking Industry Association, 2009.
- [14] A. Mendoza, **Utility Computing Technologies, Standards, and Strategies**. United States of America: Artech House, Inc., 2007.
- [15] A. Dver, "**Enemy of SaaS?,**" Software Magazine, vol. 27, no. 6, p. 24, Nov. 2008.

REFERENCIAS ELECTRÓNICAS

- [3] **Tesseract Motor OCR.** [en línea]. 27 Abril 2011. <http://en.wikipedia.org/wiki/Tesseract_%28software%29> [Consulta: 11 Mayo 2011].
- [4] **Tesseract Motor OCR.** [en línea]. 25 Noviembre 2010. <http://linuxmaniatico.blogspot.com/2010/11/ocr-en-ubuntu.html> [Consulta: 14 Mayo 2011].
- [16] **Right Scale.** [en línea]. 13 Septiembre 2010. < http://rightscale.rubyforge.org/> [Consulta: 22 Julio 2011].
- [17] **Right Scale y Web Services.** [en línea]. 25 Junio 2009. <http://en.wikipedia.org/wiki/Amazon_Web_Services > [Consulta: 22 Julio 2011].
- [18] **Right Scale/Amazon.** [en línea]. 19 Enero 2009. <http://aws.amazon.com/es/what-is-aws/> [Consulta: 22 Julio 2011].
- [19] **Amazon y GoGrid.** [en línea]. Septiembre 2010. <http://en.wikipedia.org/wiki/GoGrid> [Consulta: 23 Julio 2011].
- [20] **GoGrid Cloud Hosting.** [en línea]. Marzo 2009. <http://www.slicehost.com/questions/#users> [Consulta: 23 Julio 2011].
- [21] **Flexiscale.** [en línea]. Septiembre 2010. <http://en.wikipedia.org/wiki/FlexiScale > [Consulta: 23 Julio 2011].
- [22] **Arquitectura CloudCrowd.** [en línea]. Mayo 2010. <https://github.com/documentcloud/cloud-crowd/wiki/CloudCrowd-Architecture> [Consulta: 25 Julio 2011].
- [23] **Heroku.** [en línea]. Julio 2010. <http://about.heroku.com> [Consulta: 25 Julio 2011].
- [24] **Heroku.** [en línea]. Septiembre 2010. <http://devcenter.heroku.com/articles/config-vars> [Consulta: 26 Julio 2011].
- [25] **Heroku.** [en línea]. Junio 2010. <http://devcenter.heroku.com/articles/maintenance-mode> [Consulta: 26 Julio 2011].
- [26] **Portal Web BCV.** [en línea]. <http://www.bcv.org.ve/di> [Consulta: 14 Mayo 2012].

ANEXO: PRECISIÓN DEL OCR