PEMBUATAN APLIKASI BERGERAK TEMU ULANG FILE ELEKTRONIK BERBAHASA INDONESIA DENGAN MEMANFAATKAN JAVA CLDC

Andre Kurniawan[1], Hapnes Toba[2]
Fakultas Teknologi Informasi, Universitas Kristen Maranatha
Jl. Suria Sumantri 65 Bandung 40164; Telp. (022)2012186 ext. 296
E-mail: hackersmooth88@gmail.com; hapnestoba@yahoo.com

ABSTRACT

Smart Download is a mobile application that dedicated to search electronic documents in Indonesian (Bahasa Indonesia) throguh the Web. This application use the Advanced URL Google (spyder) to retrieve the documents from the Internet, that further will be enchanced with Porter Stemmer and the combination of TF/IDF and vector space algorithm. The documents that need to be downloaded can be breakdown in partial parts. The implementation of the application used J2ME CLDC. The application is evaluated in a classroom situation, the result shows that search is improved after the enhancement process.

Keywords: Smart Download, Spyder Google, Porter Stemmer, Vector Space Algorithm.

1. LATAR BELAKANG DAN RUMUSAN MASALAH

Masalah yang sering muncul dalam dunia *Internet* adalah bagaimana melakukan pencarian informasi yang tepat dan bagaimana strategi *download* terhadap *file* yang diinginkan secara efektif. Mesin pencarian seperti *Yahoo* dan *Google* memang dapat memberikan kandidat himpunan dokumen, lihat Lewis (2008), namun jika pencari informasi memerlukan konteks atau definisi yang terkait dengan bahasa tertentu, maka pencari informasi harus membuka dokumen yang ditawarkan satu per satu. Tentu akan sangat memerlukan waktu yang lama jika seandainya dokumen yang diinginkan ternyata ada pada urutan rendah.

Masalah lain muncul jika pencari informasi ingin melakukan *download* dari perangkat *handphone*. Sampai saat ini penggunaan terhadap perangkat lunak untuk melakukan *download* khusus seperti *flashget* masih jarang terutama di *handphone*. Biasanya pengguna *handphone* akan menggunakan *browser Opera Mini. Browser Opera Mini* hanya melakukan *download* secara *sequentional* dan tidak dapat melakukan *resume* dan *download part.*

Di dalam makalah ini akan diuraikan sebuah solusi untuk melakukan proses pencarian dokumen khususnya dalam konteks Bahasa Indonesia, dan bagaimana melakukan untuk melakukan *download file* tersebut secara efektif melalui perangkat *handphone*. Konsep yang akan digunakan adalah kombinasi antara *advanced search Google* (Spyder Google), penerapanan *text mining* untuk membantu pemilihan dokumen yang sesuai dengan bahasa Indonesia, dan *download manager* dengan menggunakan teknik *download parsial.* Aplikasi yang akan dibuat dengan menggunakan *J2ME CLDC* sehingga dapat dioperasikan dalam perangkat *handphone.*

2. LANDASAN TEORI

2.1 Teknologi J2ME

Teknologi *Java* yang pada awalnya dikenal untuk aplikasi pada *desktop (J2SE)* ataupun pada *application server (J2EE)*, kini hadir dengan teknologi terbarunya, *J2ME™ Platform*, untuk pembangunan aplikasi pada *mobile device* seperti *mobile phone* dan *PDA.* Teknologi dalam *J2ME* dibedakan berdasarkan *configuration* dan *profile.*

Configuration mendefinisikan minimum *Java Libraries* dan kapabilitas yang dimiliki oleh para developer *J2ME*. Artinya antara *mobile device* yang *Java enabled* maka akan ditemukan *configuration* yang sama. Bila dianalogikan dengan sekelompok mobil maka sebuah mobil dengan mobil lainnya memiliki kesamaan, misalnya mempunyai roda, kaca spion, spedo meter, dll. Jadi *configuration* hanyalah mengatur hal-hal yang berkaitan dengan "kesamaan" bukan mengatur hal-hal yang "membedakan", sehingga *configuration* memastikan portabilitas antar *devices*. Aplikasi *Smart Download* yang dikembangkan menggunakan *configuration CDLC*, yang dapat Digunakan pada perangkat *handheld (handphone, PDA, two way pager)* dengan memori terbatas (160-512 kB), dan prosesor 16 atau 32 bit, lihat Wirya Santika (2009).

2.2 *Advanced* URL Google

Advanced URL *Google* dibuat untuk memaksimal pencarian terhadap data yang akan ditemukan oleh pengguna. Dengan memiliki pemahaman yang baik terhadap *Advanced* URL ini diharapkan pencarian yang akan dilakukan oleh Google dapat lebih spesifik, lihat Calishain (2003).

Berikut ini adalah macam-macam operator yang dapat dikombinasikan, dan digunakan dalam aplikasi *Smart Download*, lihat Google (2009):

- *intitle*: Untuk mencari kata-kata dari judul suatu halaman *web*, misalnya *Keyword*: intitle:"java"

- *inurl*: Digunakan untuk mencari semua URL yang berisi kata-kata tertentu, misalnya *Keyword*: inurl:"java"
- *filetype*: Jika kita mencari jenis file tertentu yang berisi informasi yang anda inginkan kita bisa menggunakan operator ini, misalnya *Keyword*: "hacker" filetype:pdf

Dengan menggunakan kombinasi operator yang tepat maka pengguna akan mendapatkan informasi yang lebih relevan sesuai kata kunci yang diinginkan.

2.3 Text Mining

Salah satu bagian dari *Data Mining* yang cukup menarik adalah *Text Mining*. Metode ini digunakan untuk menggali informasi dari data - data dalam bentuk teks seperti buku, makalah, paper, dan lain sebagainya. Yang membedakan data *mining* dengan *text mining* adalah proses analisis terhadap suatu data. *Data Mining* atau KDD (*Knowledge Discovery in Databases*) adalah proses untuk menemukan pengetahuan dari sejumlah besar data yang disimpan baik di dalam *databases, data warehouses* atau tempat penyimpanan informasi lainnya. Sedangkan untuk *text mining* sering disebut dengan *Keyword-Based Association Analysis. Keyword-Based Association Analysis* merupakan sebuah analisa yang mengumpulkan *keywords* atau *terms* (istilah) yang sering muncul secara bersamaan dan kemudian menemukan hubungan asosiasi dan korelasi di antara *keywords* atau *terms* itu, lihat Tan (1999).

Secara garis besar dalam melakukan implementasi *text mining* terdiri dari 2 tahap besar yaitu *pre-processing* dan *processing.* Tahap *pre-processing* adalah tahap di mana aplikasi melakukan seleksi data yang akan diproses pada setiap dokumen. Setiap kata akan dipecah-pecah menjadi struktur bagian kecil yang nantinya akan mempunyai makna sempit. Ada beberapa hal yang perlu dilakukan pada tahap *pre-processing* ini, yaitu: *Tokenizing*, *Filtering*, dan *Stemming*. Tujuan dilakukan *pre-processing* adalah memilih setiap kata dari dokumen dan merubahnya menjadi kata dasar yang memiliki arti sempit. Kemudian tahap yang kedua adalah melakukan *processing.* Tahap ini merupakan tahap inti di mana setiap kata akan diolah dengan algoritma tertentu sehingga mempunyai bobot terhadap setiap dokumen yang akan diseleksi. Tahap ini sering disebut juga dengan *Analizing*.

Dokumen dalam bahasa Indonesia mempunyai keunikan tersendiri, karena kata - kata dalam bahasa Indonesia dapat berubah bentuk saat mendapatkan imbuhan. Untuk itu diperlukan proses s*temming*, yaitu proses mengembalikan kata ke bentuk dasar. Pada penelitian ini diterapkan algoritma *Porter Stemmer for Bahasa Indonesia*, yang dikembangkan oleh Tala (2003).

Dalam tahap *processing*, dokumen akan dianalisa oleh aplikasi. Secara umum terdapat dua jenis metode yaitu metode yang tidak melakukan perhitungan bobot kalimat dan yang melakukan perhitungan bobot kalimat. Metode yang tidak menghitung bobot kalimat hanya mengambil beberapa kalimat awal dan akhir. Metode-metode yang menghitung bobot kalimat menggunakan bobot *term* (kata maupun pasangan kata) dari setiap *term* yang terdapat dalam kalimat tersebut.

Bobot *term* diperoleh dengan melakukan perhitungan sederhana terhadap *Term Frequency* dan *Inverse Document Frequency* dari *term* tersebut yaitu *TF/IDF*. Dengan menggunakan *title factor*, bobot dari sebuah *term* dapat ditingkatkan jika *term* tersebut terdapat dalam judul. Penerapan terhadap dokumen akan dilakukan dengan menggunakan kombinasi dari Algoritma *TF/IDF* dan *Vector Space*.

2.4 Algoritma TF/IDF

Formula yang digunakan adalah menghitung bobot (W) masing-masing dokumen terhadap kata kunci, lihat Mooney (2006):

$$W_{d,t} = tf_{d,t} * IDF_t \quad (1)$$

dimana:

d = dokumen ke –d

t = kata ke – t dari kata kunci

IDF = hasil inversi dari frekuensi kata dalam dokumen, dihitung dengan mengambil nilai logaritma dari perbandingan antara jumlah dokumen yang dihasilkan dengan jumlah total frekuensi kata ke – t dari kata kunci dalam dokumen

W = bobot dokumen ke – d terhadap kata ke – t

Setelah bobot (W) masing-masing dokumen diketahui, maka dilakukan proses *sorting* dimana semakin besar nilai W maka semakin besar tingkat similaritas dokumen terhadap kata yang dicari.

2.5 Algoritma *Vector Space*

Algoritma ini merupakan algoritma kombinasi yang dilakukan setelah dokumen dihitung bobotnya dengan menggunakan algoritma *TF/IDF*. Algoritma ini diterapkan ketika bobot dari perhitungan *TF/IDF* memiliki derajat bobot yang sama antar dokumen. Algoritma ini akan mengatasi bobot yang sama untuk setiap dokumen yang akan dibandingkan.

Ide dari metode ini adalah dengan menghitung nilai *cosinus* sudut dari dua *vector* yaitu W dari tiap dokumen dan W dari kata kunci, lihat Mooney, (2006).

2.6 *Download* Parsial

Download Parsial adalah teknik *download* dengan mengirimkan *request* sebagian file dari server. Setiap server memiliki protokol yang mengatur atau mengijinkan terjadinya hubungan,

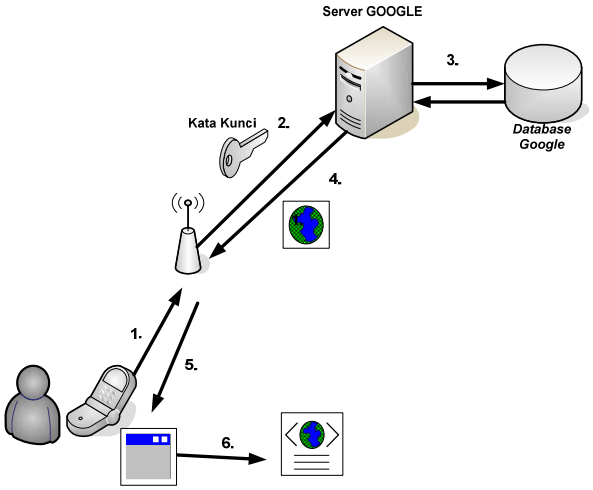
komunikasi, dan perpindahan data antara dua atau lebih titik komputer. Ketika ingin meminta informasi dari protokol maka aplikasi harus mengirimkan *request* di mana ketika server melakukan *respons* maka server akan mengirimkan informasi berupa *respons code* dan juga *stream bytes*, lihat Shackelford (2006).

3. PERANCANGAN APLIKASI

Dalam bagian ini disampaikan perancangan mekanisme pencarian dokumen, *text-mining* sampai dengan proses *download*-nya.

3.1 Mekanisme Pencarian Dokumen

Gambar 1 memperlihatkan mekanisme pencarian dokumen.



Gambar 1. Mekanisme Pencarian Dokumen

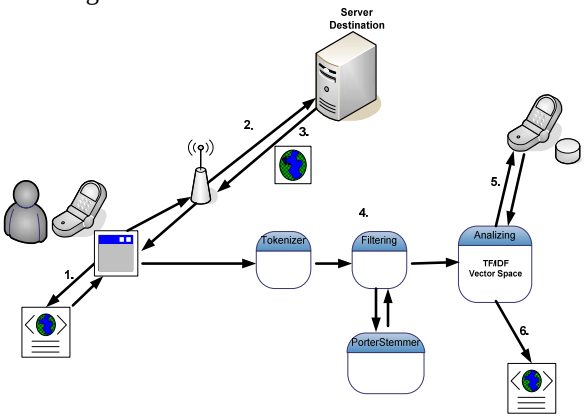
1. Aplikasi akan mengirimkan *Header HTML URL* di mana aplikasi melakukan ini dengan membuka *port 80*. Dalam hal ini aplikasi harus terhubung dengan *Internet.*
2. Kata kunci akan dikirimkan ke *Google* dengan melakukan kombinasi.
3. *Google* akan meresponse *HTML REQUEST* ini dan mengecek ke dalam d*atabase*-nya
4. Halaman yang diminta berhasil didapatkan dan *Google* akan mengirimkan hasilnya.
5. Aplikasi akan menangkap *Tag HTML*.

Aplikasi akan menggunakan menyimpan informasi ini ke dalam format *XML*. Kemudian data akan diolah sesuai dengan kategori. Pada tahap ini aplikasi belum melakukan *mining* hanya melakukan perbandingan *hit point.*

3.2 Mekanisme *Text-Mining*

Gambar 2 memperlihatkan mekanisme *text-mining*.

1. Data yang sudah ditampilkan ketika dilakukan pencarian *file* pada fitur sebelumnya. Data akan dikelola dalam bentuk *XML* dan siap untuk dipilih di dalam menu *search*.
2. Data yang sudah dipilih dari *Choice Group* yang ada pada menu *search* ini akan di-*download* dalam bentuk *HTML*. Hal ini dilakukan agar data dapat di-*parsing* dengan baik oleh aplikasi dan tidak membuang *bandwith* karena berukuran kecil, dalam bentuk XML.
3. Data yang telah diambil akan diubah kembali menjadi *HTML* dan siap untuk diolah oleh aplikasi.
4. Data yang sudah disimpan ke dalam aplikasi maka akan segera diolah di dalam modul *mining*. Data tersebut akan melewati 4 proses *text mining* antara lain, yaitu:
   - *Tokenizing*
   - *Filtering*
   - *Stemming*
   - *Analizing* yang menerapkan algoritma *TF/IDF* dan *Vector Space*
5. Aplikasi akan melakukan *link associate* di mana akan menjadi setiap *log* yang sudah dilakukan oleh pengguna. Hal ini dilakukan untuk mendapatkan pencarian yang mendekati keinginan *user*.

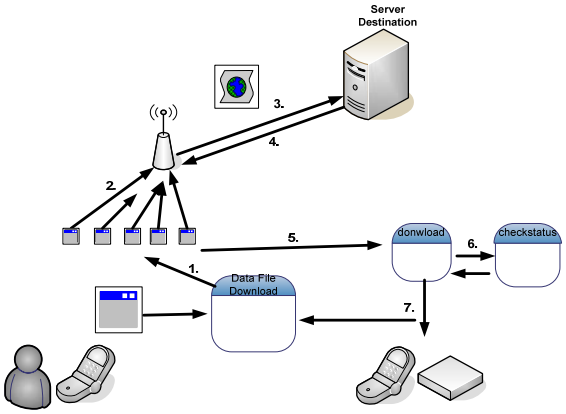


Gambar 2 Mekanisme Text-Mining

3.3 Mekanisme Download Parsial

Gambar 3 memperlihatkan rancangan mekanisme untuk *download file*:

1. Aplikasi akan melakukan *load* terlebih dahulu. Data setiap informasi mengenai *file* akan disimpan di dalam *Record Store*
2. Setelah data berhasil di-*load* maka aplikasi akan melakukan *thread* sesuai dengan pembagian *job* yang sudah diatur dengan melakukan *download* berdasarkan ukuran *byte*. Kemudian setiap *thread* akan melakukan koneksi terhadap *server* tujuan
3. Setiap *job* akan mengirimkan *head* yang berbeda-beda. Dan pada tahap ini aplikasi dapat meminta *server* melakukan *request file* secara parsial. Tahap ini juga menentukan *download manager* untuk melakukan *download file* secara parsial.



Gambar 3. Mekanisme *Download* Parsial

4. Apabila *protokol server* mendukung metode *partial*, maka *server* akan mengirimkan *file* sesuai dengan *byte* yang diminta oleh *user*.
5. Setiap *byte* yang akan diterima akan disimpan terlebih dahulu dalam bentuk *stream.* Kemudian aplikasi akan melakukan pengecekan terhadap modul *download*.
6. Setiap kali melakukan *download* maka aplikasi akan mengecek status *job* apakah mati atau tidak. Hal ini untuk mengantisipasi apabila terjadi *pause job*.
7. Aplikasi akan memindahkan *byte* yang sudah disimpan ke dalam *file handphone.* Kemudian akan melakukan *update* terhadap data *file*.

3.4 Diagram Kelas

Jumlah keseluruhan kelas yang membentuk aplikasi *Smart Download* ini adalah 31, dapat dilihat pada Lampiran 1. Dalam gambar diagram kelas terdapat dua *package* yang dilingkari besar. Hal ini untuk menunjukkan bahwa kelas tersebut mengambil beberapa *library* tambahan yang diambil dari *sourgeforge.net*, yaitu:

1. FTP
   MEFTPClient, MEFTPClientController, dan *MEFTPClientDataConnection.*
2. XML
   XML, ParseEvent, XMLParser, dan *AbstractXMLParser*

Setiap pencatatan konfigurasi dan informasi mengenai *file* akan dicatat ke dalam internal memori menggunakan Kelas *RecordData.* Kelas *GoDownload* adalah kelas yang membuat objek untuk menjalankan *download manager*. Dari kelas inilah implementasi *multithreading* akan dilakukan pada saat pemecahan *file*.

Kelas *Abstract Search* akan menangani koneksi ke dalam *Google* dan melakukan *download* terhadap *file* yang akan diproses. Proses *mining* akan dilakukan oleh kelas *GoMining.* Kelas ini juga akan melakukan implementasi dan sinkronisasi antar kelas *ParseXML* dan *package mining.* Untuk kelas seperti *HTTP, FTP, MemoriMonitor*, dan lain lain akan mendukung proses pada masing–masing modul secara fungsionalitas yang spesifik tugas tertentu.

4. HASIL IMPLEMENTASI

4.1 Pencarian Dokumen

Pada modul ini dilakukan dua macam pengolahan data dalam melakukan pencarian *file* yaitu:

- ***File Html/Page*** → Pada tahap ini *file html* yang didapatkan dari *Google* akan disimpan ke dalam *XML*. Data yang akan dikirimkan ke dalam *URL Google* akan dikombinasikan dengan kata kunci khusus *Google.* Banyaknya informasi *page* yang akan di-*download* tergantung dari pengguna yang akan melakukan *set* pada menu *setting*.
- ***Hit point*** → Tahap ini merupakan tahap lanjutan dari tahap pengambilan informasi dari *Google.* Tahap ini belum melakukan implementasi *mining* tetapi melakukan *filtering* data dengan melakukan perbandingan terhadap 3 himpunan hasil pencarian dari *Google* dengan operator: *filetype:doc,inurl:,intitle:*.

Tiga hasil ini akan digunakan untuk melakukan perbandingan. Tahap ini akan melakukan seleksi terhadap halaman *Google* yang telah dikelola di dalam XML. Kemudian akan membandingkan setiap *link* yang muncul apakah ada ketika melakukan kombinasi. Setiap *URL* yang ada di halaman *Google* akan diberikan nilai hit sesuai dengan urutannya.

Contoh: untuk sebuah dokumen D1, diperoleh hasil untuk setiap operator di atas menghasilkan:
MAXPAGE (jumlah halaman pencarian maksimum) = 20

filetype*:* Posisi 8, nilai: [MAXPAGE] –8
inurl: Posisi 3, nilai: [MAXPAGE] – 3
intype: Posisi 5, nilai: [MAXPAGE] – 5

Total Hitpoint D1: [(20-8) + (20-3) + (20-5)] / [MAXPAGE*3] →**73.3%**

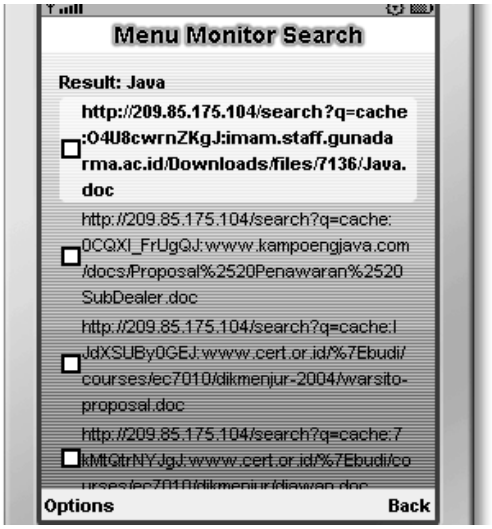
Hit point dari setiap dokumen yang akan ditampilkan tergantung dari presentase minimal yang diinginkan oleh pengguna. Apabila pengguna mengisi presentase 70% maka contoh D1 di atas akan keluar pada halaman hasil. Setelah data sudah didapatkan maka aplikasi baru dapat melakukan *mining* terhadap data yang akan diseleksi. Pengguna tinggal memilih dengan melakukan *check* pada *form.*

Gambar 4, memberikan hasil implementasi untuk pencarian file sebelum dilakukan *text-mining*.

4.2 Text-Mining

Di bawah diberikan contoh dokumen yang akan diseleksi dan perhitungan dalam matriks pengolahan:
Kata Kunci(kk) = pengetahuan logistik
Hasil:
Dokumen 1 (D1) = Manajemen transaksi logistik
Dokumen 2 (D2) = Pengetahuan antar individu

Dokumen 3 (D3) = Dalam manajemen pengetahuan terdapat transfer pengetahuan logistik.
Jumlah dokumen ditemukan (D)=3



Gambar 4. Hasil Pencarian File

Setelah melalui proses *filtering* dan *stemming*, maka kata "antar" pada dokumen kedua serta kata "dalam" dan "terdapat" pada dokumen ketiga dihapus. Tahap pembuangan ini akan dilakukan pada proses filtering menggunakan algoritma *stoplist* (membuang kata yang tidak penting).

Dari hasil pada Gambar 5, yang diimplementasikan dengan matriks 2*2, akan dapat dihitung bobot W, tapi masalah yang terjadi adalah nilai bobot untuk D1 dan D2 adalah sama**.** Untuk mengatasi hal ini digunakan algoritma *Vector – Space*. Ide dari metode ini adalah dengan menghitung nilai kosinus sudut dari dua vektor, yaitu: W dari tiap dokumen dan W dari kata kunci.

Kemudian dilakukan perhitungan lagi ke dalam matriks 2*2, sebagaimana hasilnya terlihat dalam Gambar 6.

	tf				df	D/df	IDF	W			
token	kk	D1	D2	D3			log(D/df)	kk	D1	D2	D3
manajemen	0	1	0	1	2	1.5	0.176	0	0.176	0	0.176
transaksi	0	1	0	0	1	3	0.477	0	0.477	0	0
logistik	1	1	0	1	2	1.5	0.176	0.176	0.176	0	0.176
transfer	0	0	0	1	1	3	0.477	0	0	0	0.477
pengetahuan	1	0	1	2	2	1.5	0.176	0.176	0	0.176	0.352
individu	0	0	1	0	1	3	0.477	0	0	0.477	0
Total	0.352	0.829	0.653	1.181							

bobot (*w*) untuk D1 = 0.176 + 0 = 0.176
bobot (*w*) untuk D2 = 0 + 0.176 = 0.176
bobot (*w*) untuk D3 = 0.176 + 0.352 = 0.528

Gambar 5 Contoh Hasil Perhitungan Bobot Antar Dokumen

Setelah mendapatkan nilai maka digunakan persamaan 1, misalnya untuk D3:

cos(D3)=sum(kk*D3)/[sqrt(kk)*sqrt(D3))]
= 0.093/[0.249 * 0.643]
= 0.581

token	kk	D1	D2	D3	Kk*D1	Kk*D2	kk*D3
manajemen	0	0.031	0	0.031	0	0	0
transaksi	0	0.228	0	0	0	0	0
logistik	0.031	0.031	0	0.031	0.031	0	0.031
transfer	0	0	0	0.228	0	0	0
pengetahuan	0.031	0	0.031	0.124	0	0.031	0.062
individu	0	0	0.228	0	0	0	0
	Sqrt(kk)	Sqrt(Di)			Sum(kk dot Di)		
	0.249	0.539	0.509	0.643	0.031	0.031	0.093

Gambar 6 Perhitungan Bobot menggunakan *Vector Space*

Dari hasil perhitungan di atas, maka akan didapatkan:

	D1	D2	D3
Cosine	**0.231**	**0.245**	**0.581**
	Rank 3	Rank 2	Rank 1

Untuk memaksimalkan algoritma TF/IDF dan *Vector Space* maka diterapkan derajat asosiasi. Derajat asosiasi adalah proses pencarian kata dari tiap dokumen yang memiliki relevansi terhadap kata kunci. Jadi setiap kata kunci yang akan dimasukkan sistem akan dicatatkan *history*-nya ke dalam *HashTable* yang disimpan di dalam RMS Chakrabarti (2003). Kemudian apabila ada kata kunci yang akan dicari memiliki relevansi maka kata yang ada di dalam *HashTable* akan diikutsertakan di dalam perhitungan bobot *TF/IDF*.

Apabila pengguna melakukan pencarian dengan kata kunci "hacker" maka aplikasi akan mencatat kata–kata apa saja yang masuk ke dalam perhitungan bobot, lihat Dhillon (2001).

Contoh:

LOG 1:

Kata yang ikut perhitungan bobot TF/IDF → hack,intruder

LOG 2:

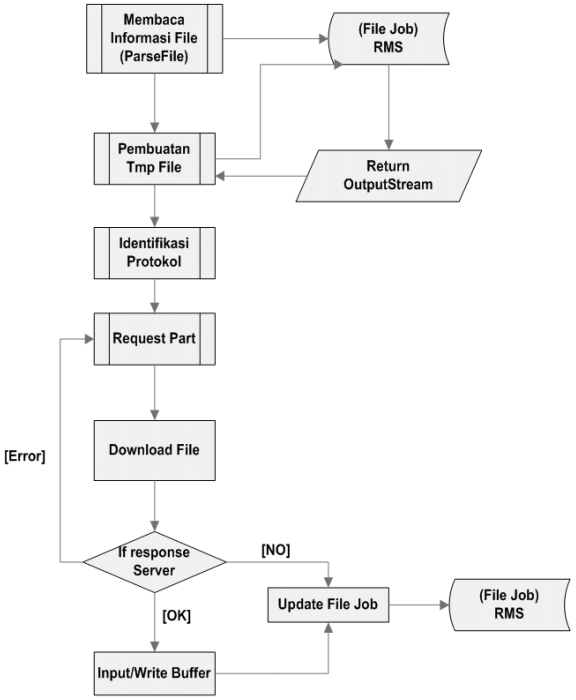
Kata yang ikut perhitungan bobot TF/IDF → nmap, scan, Ethical

Maka Aplikasi akan merubah menjadi: nmap, scan, Ethical, hack, intruder

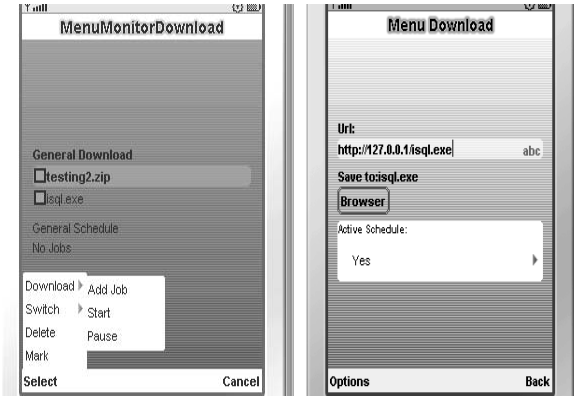
Jadi kata yang memiliki kesamaan kata kunci akan diikutsertakan. Hal ini akan menjadikan penambahan bobot sehingga dokumen dapat benar – benar diseleksi karena perbendaharaannya yang diatur pada derajat asosiasi.

4.3 Download Manager

Gambar 7 menunjukkan proses kerja *Byte Streaming* dan juga *Prokotol* untuk pemecahan file. Setiap informasi akan dicatat ke dalam *RMS*. Gambaran hasil tercapai dapat dilihat pada Gambar 8.



Gambar 7 *Flowchart* Pemecahan *File*



Gambar 8. Realisasi Download pada Form J2ME

5. EVALUASI

Evaluasi dilakukan terhadap 28 responden mahasiswa kelas Basis Data G Teknik Informatika UK. Maranatha semester ganjil 2008/09, untuk melihat performa aplikasi *Smart Download.*

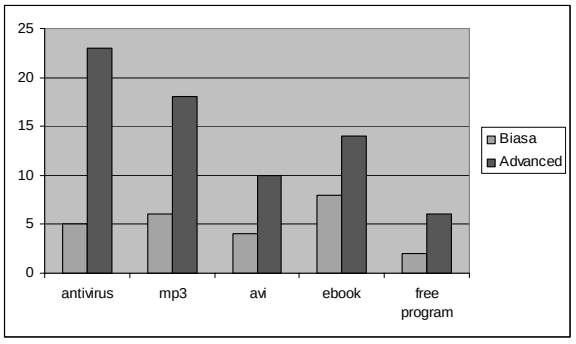
Dalam pengujian *Black Box* yang diuji adalah menu utama, yaitu: Menu *Home*, Menu *Search*, Menu *Download*, Menu *Bookmarks*, dan Menu *Option*. Uji kasus dibuat dengan tiga macam skenario yang berbeda. Skenario yang digunakan adalah:

1. Fungsionalitas *Advanced Search*

Pada tahap ini pengujian dilakukan dengan melakukan pencarian terhadap kata kunci: *antivirus*, *mp3*, *avi*, *ebook*, dan *free program* dengan banyak dokumen 30. Gambar 9 menunjukkan hasilnya. Hasil Google akan dibandingkan dengan hasil pencarian dari aplikasi dengan melihat berapa halaman yang berubah posisi menjadi lebih baik. Terlihat bahwa dengan penggunaan operator *Advanced Search*, hasil pencarian menunjukkan jumlah dokumen yang spesifik terhadap kata kunci.

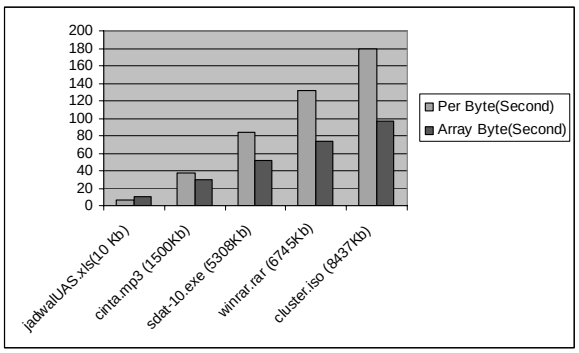
2. Fungsionalitas *Download Manager* Terhadap Protokol

Pengujian dilakukan pada *emulator* dan *gadget* (*handphone* sungguhan). Dari segi fungsionalitas aplikasi dapat berjalan dengan baik. Hanya untuk melakukan penulisan dan koneksi untuk *gadget*, terhalang oleh *certificate* yang tertanam di dalam *handphone*, baik untuk merek Sony Erickson maupun Nokia yang berbasis Java .



Gambar 9. Pengujian *Advanced Search*

Dengan *certificate* yang asli, aplikasi akan mempunyai otoritas terhadap *security internal*. Untuk mencoba fungsionalitas dari *Download Manager*, maka melakukan percobaan dengan melakukan akses dengan protocol *HTTP*, *HTTPS*, dan *FTP*.



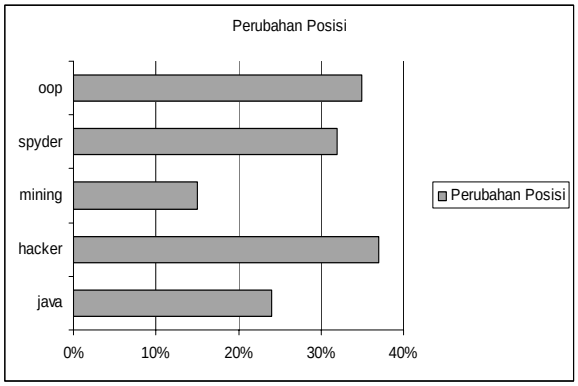
Gambar 10. Pengujian Protokol dan Byte

Pada Gambar 10, terlihat bahwa performa pemecahan file dengan *byte stream* lebih baik, dan terbukti dapat digunakan untuk mengambil dokumen dari server dengan protokol yang berbeda-beda.

3. Fungsionalitas *Mining* dengan dokumen yang beragam

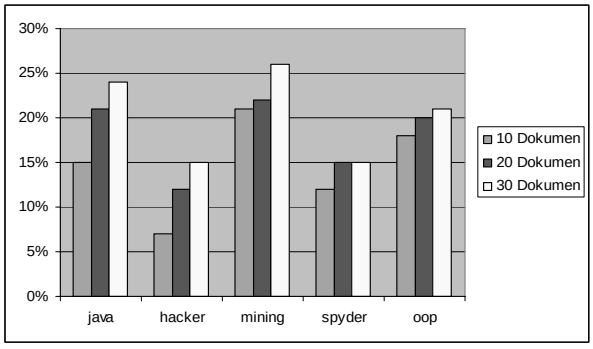
Untuk melakukan fungsionlitas dari *text mining* ini maka pengujian dilakukan ke dalam 2 skenario yaitu:

- **Perbandingan dengan *Google***
  Skenario yang digunakan adalah dengan melakukan seleksi terhadap 30 dokumen. Kata kunci yang digunakan adalah java, *hacker*, *mining*, *spyder*, dan OOP.
  Pada Gambar 11, terlihat bahwa terjadi perubahan posisi (relevansi) yang terjadi, setelah dilakukan *text-mining*.
- **Relevansi yang terjadi dengan banyak dokumen**
  Skenario yang dibuat adalah dengan mengunakan kata kunci seperti pada skenario sebelumnya kemudian mencoba melakukan uji coba dengan seleksi terhadap dokumen non-Indonesia.



Gambar 11. Perubahan Posisi Setelah *Text Mining*

Pada Gambar 12, terlihat bahwa relevansi hasil *text-mining* dengan kata kunci tidak dipengaruhi oleh banyaknya dokumen yang diseleksi.



Gambar 12. Relevansi dengan Banyak Dokumen

6. KESIMPULAN DAN SARAN

Kesimpulan yang dapat ditarik dari hasil evaluasi yaitu secara umum aplikasi ini menghasilkan nilai guna yang cukup tinggi. Aplikasi ini memberikan solusi pada masalah pencarian *file* berbahasa Indonesia dan juga dapat melakukan *download file* pada *handphone.* Fitur *mining* yang diberikan oleh aplikasi dianggap berguna oleh pengguna untuk membantu melakukan pemilihan terhadap data yang akan diambil. Presentase ketepatan *text-mining* dapat lebih dimaksimal lagi dengan melakukan seleksi terhadap setiap kata yang ada di dalam dokumen, kemudian dihitung asosiasinya dengan kata kunci.

Untuk memperbaiki performa aplikasi, dapat disarankan hal-hal sebagai berikut:

- Penerapan algoritma *mining* dapat dipindahkan dari *handphone* ke dalam *server* dengan menjalankan fasilitas *web services* yang ada pada *Java.*
- Penambahan *help* dan *user* manual yang digunakan untuk aplikasi
- Aplikasi dapat melakukan pendataan *user* siapa saja yang sudah melakukan pencarian dokumen dan mencatat setiap *log* sesuai kebutuhan *user*, sehingga dapat membentuk *collaborative searching.*

PUSTAKA

Chakrabarti, Soumen, 2003, *Mining the Web: Discovering knowledge from hypertext data.* San Francisco: Morgan Kaufman

Calishain, Tara. 2003. *Google Hack.O'Reilly*

Dhillon, Inderjit S., Dharmendra S. Modha, 2001, *Concept Decompositions for Large Sparse Text Data using Clustering, Machine Learning, vol. 42, no. 1, pp. 143-175, January 2001*

Google, Advanced Google Search Operators, Retrieved February 2nd, 2009 from http://www.google.com/help/operators.html

Mooney, Raymond. 2006. *Machine Learning Text Categorization*. Austin: University of Texas

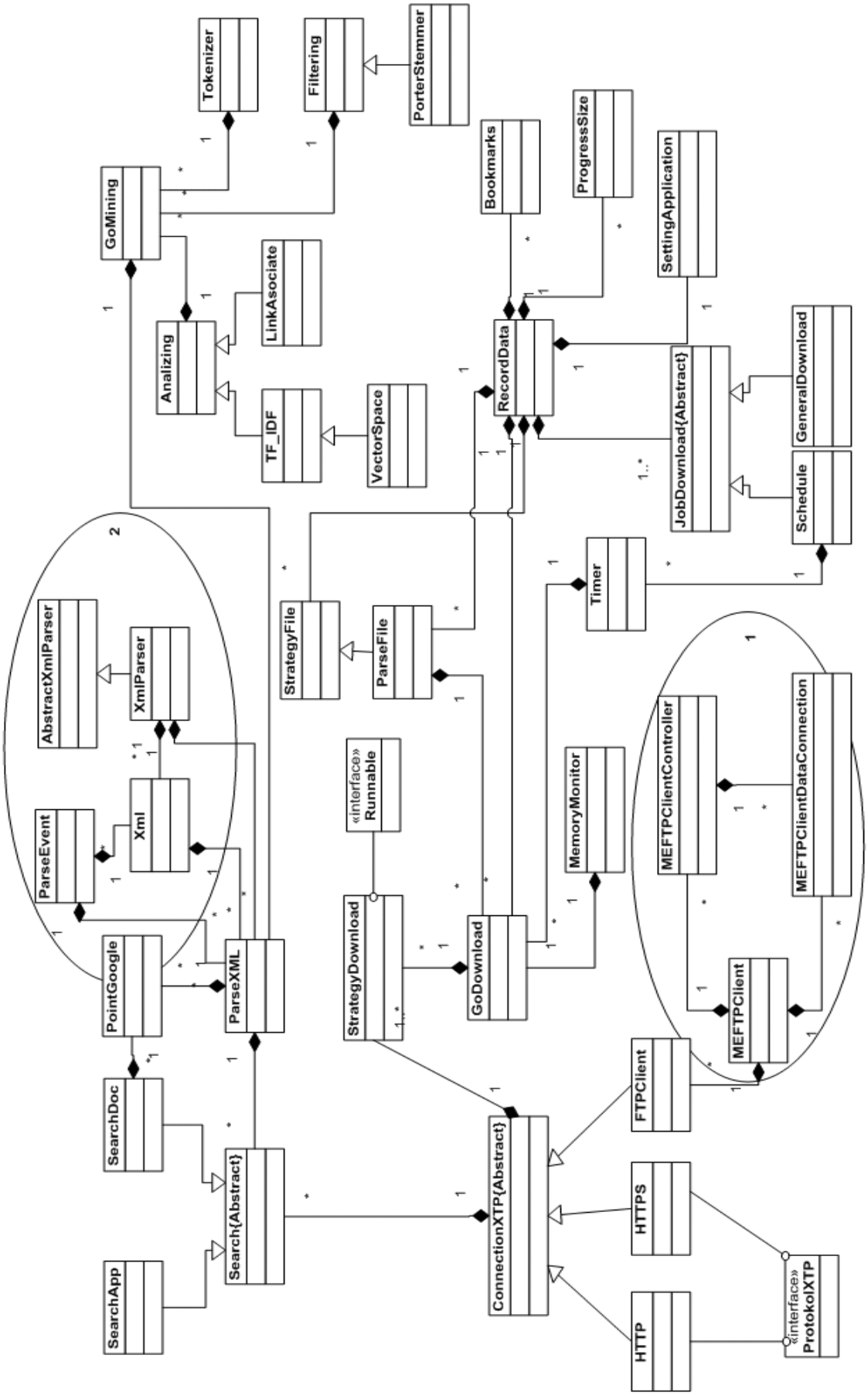
Tala, Fadillah Z., 2003, *A Study of Stemming Effects on Information Retrieval in Bahasa Indonesia. Institute for Logic*, Language and Computation Universeit Van Amsterdam

Tan, A., 1999, *Text mining: The state of the art and the challenges*, In Proceedings of the Pacific Asia Conference on Knowledge Discovery and Data mining, *PAKDD'99 workshop on Knowledge Discovery from Advanced Databases.p.65-70*

Porter, M. F. ,1980, *An algorithm for suffix stripping, Program 14(3), p. 130-137*

Lewis, Edward. 2008. *Top 10 Search Engine.* Retrieved January 5th, 2008 from *http://www.seoconsultants.com/search-engines/*

Wirya Santika, Faisal. 2009. *Membangun Wireless Application Menggunakan Teknologi J2ME.* Retrieved January 5th, 2009 from http://ilmukomputer.org/2008/11/25/membangun-wireless-application-menggunakan-teknologi-j2me/



Lampiran 1. Rancangan Kelas Diagram