Projekt rrs_page_change_monitoring

1. Požadavky na systém, use cases, koncepce

TODO: Tuto kapitolu bude třeba nějak restrukturalizovat a udělat pořádně. (udělat UCs a nějaké malé FAQ).

1.1 Požadavky

1. možnost zjistit verzi dokumentu např. před měsícem | use cased (UC1)
2. možnost zjistit aktuální verzi dokumentu | use cased (UC2)
3. možnost zjistit, zda byla stránka od posledního checku změněna | use cased (UC3)
4. jaké byly změny od posledního checku dokumentu? | use cased (UC4)
5. jaké byly změny od určitého času T1 do času T2? | use cased (UC5)
6. možnost přinutit systém, aby zkontroloval danou URL (check) | use cased (UC6)
7. možnost zjistit, zda je požadovaná URL dostupná | use cased (UC7)
8. možnost zjistit, zda byla URL dostupná, když byla naposledy kontrolována -- rozšířeno na předání celé historie | use cased (UC8)
9. možnost zjistit poslední DOSTUPNOU verzi dokumentu (bude se používat když UC2 failne) | use cased (UC9)
10. možnost kontrolovat jak webové stránky, tak i jiné dokumenty | s tím počítáme, máme rozmyšlené i binární diffy, http je to jedno
11. možnost omezení velikost sledovaného souboru (aby se nemuselo do db ukládat moc dat) | !zanést do use case detailů!
12. musí být naprosto obecný, nespecializovaný pro jakýkoliv účel (pro nějaký projekt) | toho se držíme
13. vyrovnat se s “url shortening” - bitly.com, goo.gl (2.1.1) | neni problém
14. vyrovnat se s redirecty (301,302,303) a cookies | řešíme (2.1.2)
15. ??

1.2 UCs

UC	**Template**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1.
	2. KDYŽ URL <UC končí>
	3.
Výstupní podmínky	
Poznámka	

Názvosloví:

- *check* - událost, kdy se uživatel systému dotáže na danou url. Pokud byl změněn obsah od posledního checku, systém obsah stáhne.
- *dokument* - jeden soubor libovolného druhu. Uživatel dokumenty identifikuje pomocí URL. (fakt, že jeden dokument může být dostupný přes různé URL je řešen interně a uživatel o tom nesmí vědět)
- *dostupná URL* - URL je dostupná, pokud pro ni server navrací response code < 400
- *čas verze dokumentu* - údaj z HTTP hlavičky “last-modified”. POZOR! Na last-modified se nedá spoléhat jako arbitr o změně, některé servery odpovídají v last-modified aktuálním časem!
- *čas checku* - timestamp, kdy byla přijata HTTP hlavička daného dotazu

UC1	**Možnost zjistit verzi dokumentu v daném čase v minulosti**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1. KDYŽ dokument (pod zadanou URL) není uložen v databázi, systém hlasí chybu. <UC končí>
	2. KDYŽ v databázi neexistuje verze starší, než je dotazovaný čas, systém navrátí nejstarší možnou verzi. <UC končí>
	3. KDYŽ je dotazovaný čas novější, než čas nejnovější verze v db, systém použije UC6, tím zajistí aktuálnost DB vzhledem k danému času. Případná chyba v rámci UC6 bude propagována až k uživateli a UC skončí.
	4. Systém vybere verzi dokumentu, jejíž doba platnosti pokrývá zadaný čas (tzn. nejmladší, která byla vytvořena před žádaným časem)
Výstupní podmínky	Rozhraní poskytlo uživateli obsah URL v žádaném čase (a čas jeho vytvoření).

UC2	**Možnost zjistit aktuální verzi dokumentu**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1. Systém vyvolá interně UC6 (check) na uživatelem danou URL
	2. KDYŽ URL není dostupná, systém hlásí error <UC končí>
	3. Systém vybere nejnovější verzi dokumentu z DB.
Výstupní podmínky	Rozhraní poskytlo uživateli aktuální obsah URL.

Poznámka	Funkčně se kryje s UC1(url, now)

UC3	**Možnost zjistit, zda byla stránka od posledního checku změněna**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1. Systém interně vyvolá UC6 (check) na uživatelem danou URL
	2. KDYŽ URL není dostupná, systém hlásí error <UC končí>
	3. Systém porovná nově získanou HTTP hlavičku s poslední, kterou získal při práci pro tohoto uživatele. Pokud hlavička neobsahuje dostatečné informace, je třeba stáhnout celý obsah a porovnat jej.
Výstupní podmínky	Rozhraní poskytlo uživateli informaci o tom, zda byl dokument od posledního uživatelova checku změněn.
Poznámka	

UC4	**Jaké byly změny od posledního checku dokumentu?**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1. Systém interně vyvolá UC6 (check) na uživatelem danou URL
	2. KDYŽ URL není dostupná, systém hlásí error <UC končí>
	3. KDYŽ vyvolaný UC6 stáhl hlavičku indikující změnu dokumentu proti poslednímu checku konanému pro tohoto uživatel, vrátí systém diff verze odpovídající poslednímu checku tohoto uživatele a verze aktuální.
	3. JINAK vrátí diff odpovídající nulové změně.
Výstupní podmínky	Rozhraní poskytlo uživateli rozdílový soubor od posledního checku.
Poznámka	Posloupnost UC3, UC4 vrátí prázdnou změnu.

UC5	**Jaké byly změny od určitého času T1 do času T2?**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))

Tok událostí	1. Systém interně vyvolá UC1 pro časy T1 a T2
	2. KDYŽ některý z těchto způsobí chybu, je ta propagována uživateli <UC končí>
	3. Systém vyrobí diff souborů získaných z kroku 1 a vrátí jej uživateli
Výstupní podmínky	Rozhraní poskytlo uživateli rozdílový soubor verzí dokumentu v časech T1 a T2.
Poznámka	

UC6	**Možnost přinutit systém, aby zkontroloval danou URL (check)**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1. Systém získá informace o současné verzi dokumentu na dané URL.
	2. Systém aktualizuje informaci o tom, kdy URL naposledy kontroloval
	3. KDYŽ dokument není dostupný, systém hlásí chybu <UC končí>
	4. KDYŽ se dokument od posledního checku změnil, systém stáhne jeho aktuální verzi a informuje uživatele, že dokument byl změněn
	4. JINAK pokud je dokument dostupný systém informuje uživatele, že se dokument na dané URL od posledního checku nezměnil
Výstupní podmínky	V DB systému je verze dokumentu aktuální v okamžik volání UC, čas checku a celá získaná hlavička (tj. vč. informace o dostupnosti).
Poznámka	Před uložením souboru do DB zkontroluje jeho velikost. **@Standa**: Oznámíme uživateli, že jsme soubor neuložili? **@Karel:** No určitě, dokonce tam bude výjimka, něco jako DocumentTooLarge. Ale uživatel bude mít možnost pomocí parametru force toto obejít.

UC7	**Možnost zjistit, zda je požadovaná URL dostupná**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1. Systém interně vyvolá UC6 (check), tím se do databáze uloží informace o aktuálním stavu vč. dostupnosti.
	2. Systém, na základě informaci v DB, zpraví uživatele o tom, zda je URL aktuálně dostupná.

Výstupní podmínky	Rozhraní poskytlo uživateli informaci o tom, zda je URL aktuálně dostupná.
Poznámka	

UC8	**Možnost zjistit, zda byla URL dostupná, když byla naposledy kontrolována**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1. KDYŽ daná URL není v databázi, systém ohlásí chybu <UC končí>
	2. Systém vrátí všechny dvojice (čas checku, dostupnost ano/ne), které jsou k dané URL v databázi
Výstupní podmínky	Rozhraní poskytlo uživateli historii dostupnosti dané URL.
Poznámka	Rozšířil jsem “funkcionalitu” tohoto UC tak, aby vracel celou historii. (Karel)

UC9	**Možnost zjistit poslední DOSTUPNOU verzi dokumentu**
Vstupní podmínky	Uživatel zná dotazovanou URL (samozřejmě :))
Tok událostí	1. Systém vnitřně vyvolá UC6 na danou URL
	2. KDYŽ pro danou URL není v databázi žádný dokument, systém ohlásí chybu <UC končí>
	3. Systém vrátí nejnovější dokument příslušný danému URL
Výstupní podmínky	Rozhraní poskytlo uživateli poslední dostupnou verzi dokumentu
Poznámka	

1.3 Koncepce systému

Náš systém sestává z:

- linkcheckeru
- webového archivu (DB)
- (knihovního) rozhraní

Náš systém je tady pro:

- každého :) (libovolný projekt v rámci NLP)

Náš systém umí:

- říct, zda byl dokument na dané URL od dd.mm.yy(.hh:mm:ss) změněn
- poskytnout uživately tyto změny (v podobě nějakého diffu)
- říct, zda je stránka dostupná (vč. “historie dostupnosti”)
- odpovídat na otázky “změnilo se něco od té doby, co jsem tady JÁ byl naposled?”

Druhy dotazů na systém:

- **check** - základní dotaz pro kontrolu URL
- **get_version** - dotaz pro zjištění nějaké verze (pořadí, čas) dokumentu
- **get_diff** - dotaz pro zjištění rozdílu mezi verzemi

Pokud zjistíme, že byl dokument změněn, automaticky stáhneme do webového archivu jeho novou podobu. Tímto přenecháváme periodu aktualizace timeline daného dokumentu plně v rukou uživatele. Předpokládáme totiž, že uživatel je informovaný o dynamice daného dokumentu lépe, než my.

1.3.1 Otevřené otázky

Linkchecker

- používat vždycky HEAD a k tomu někdy GET, nebo někdy HEAD a někdy podmíněný GET?
- omezit počet dotazů na jednu url na třeba 1 / 10sec ?
- V jaké podobě si pamatovat jméno tazatele? V té podobě, jakou si vybere (ale samozřejmě pro rrs to bude jméno modulu (rrs_deliverables, rrs_cokoliv). Dáme tam funkci na ověření, jestli to jméno už někdo používá (to samozřejmě bude jen orientačně na začátku, protože po prvním vložení dat z toto jméno “už to jméno někdo bude používat - já”).

Webový archiv

- ukládat diffy, nebo snapshoty? Pokud použijeme nosql, tak potom snapshoty.

Rozhraní

- V jaké podobě vracet data, když si o ně uživatel řekne?
- Měl by check_page(url) vůbec vracet data, nebo by to mělo být odděleno?
- A spousta dalších věcí, které vyplývají ze stavby DB...

1.3.2 Diskuze

Asi tam bude něco jako MonitoredURL/MonitoredResource, což bude nějaká abstrakce internetového zdroje, který má obsah WebContent, který bude mít už přístup k těm změnám.. A asi check_page bude vracet MonitoredResource...

2. Poznámky k problému (analýza, orientace)

2.1 Zjišťování změny dokumentu

- je potřeba nastudovat **HTTP hlavičky** pro tento účel - http://odino.org/don-t-rape-http-if-none-match-the-412-http-status-code/ a http://stackoverflow.com/questions/998791/if-modified-since-vs-if-none-match
- a také obecná znalost polí **http response** - http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

- **url shortening**: http://en.wikipedia.org/wiki/URL_shortening - možnost, že jeden obsah je přístupný z různých URL adres (url adresa je většinou zkrácena a zakódována do hashe).
- zkontrolovat, jestli když server odpoví 304 Not Modified, tak jestli posílá jiný e-tag (asi to bude ve specifikaci, ale také to chce testnout na reálném serveru jestli to fakt dělá). Podle RFC 2616 (https://tools.ietf.org/html/rfc2616#section-10.3.5) ETAG vůbec poslat nemusí.
- zkontrolovat, jestli MD5 hash obsahu co vrací server (pokud jej vrací -- a ony tak činí poměrně zřídka), sedí s tím, co spočítáme. Pokud ne, vyhledat chybu - tzn zjistit, jestli je chyba na naší straně (tím jak to počítáme) nebo jestli je to nějaký starý MD5 hash na serveru, co já vím, co můžou ty servery posílat za blbosti..

Postup při ověřování, že je dokument na URL změněn (algoritmus):

1. Pošli HEAD request nebo GET s etagem, if-modified-since a if-none-match, pokud je vyžadován obsah, jinak HEAD bez e-tagu, bez ničeho...)
2. Pokud response_code == 304 (Not Modified), pak updatni “checked” u posledního http headeru pro daný link. Atribut “checked” ma semantiku “naposledy jsem to checknul dd.mm.yy.hh:mm:ss”.
3. Pokud response_code == 200 nebo byl proveden úspěšný redirect (301-303), pak se zkontroluje, jestli má obsah stejnou délku (content-length) a pokud ano, tak i md5 hash (pokud je v response headeru). Pokud bude vše stejné, pak je to stejné jako ve 2.) a pokud to na něčem selže, pak se pošle GET na daný soubor, vytvoří nový http header a diff. Pokud není md5 v response headeru, tak se soubor stáhne a ověří se MD5 hash.
4. Pokud response_code >= 400 (chyby, server errory), pak se uloží nový http header.

2.1.1 URL shorthening

Testováno pythoními funkcemi ve tvaru:

```
 7 def url_response_code(url_text):
 8     url = urlparse.urlparse(url_text)
 9
10     h = httplib.HTTPConnection(url.netloc)
11
12     params = urllib.urlencode({})
13     headers = {}
14     h.request("HEAD", url.path, params, headers)
15
16     response = h.getresponse()
17     return response.status
18
19 def url_headers(url_text):
20     url = urlparse.urlparse(url_text)
21
22     h = httplib.HTTPConnection(url.netloc)
23
24     params = urllib.urlencode({})
25     headers = {}
26     h.request("HEAD", url.path, params, headers)
27
28     response = h.getresponse()
29     return response.getheaders()
```

Testovací data

Pouze adresy tyto tři adresy (předpokládám, že URL shorthenery jsou střízlivé)

- https://docs.google.com/document/d/1eu8QP4I-r0FPxbxbYy5OI1cxj82kcGtB28L2ythpmKQ/edit?ndplr=1#
- http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

- http://stackoverflow.com/questions/9019134/httpconnection-request-socket-gaierror-in-python

Výsledky

URL shorthener	Status code	Header 'location'
bit.ly	301	Korektní
goo.gl	301	Korektní
TinyURL.com	301	Korektní

Závěr
URL shorthening problém není, všechny tři dominantní URL shorthenery se chovají velmi přívětivě.

2.1.2 Přesměrování a cookies

Experimentálně jsem implementoval pythoní funkci URL -> URL, která sleduje přesměrování (HTTP Status code in [301,302,303]).

V zásadě to funguje, kupř. bez problémů projde přes bit.ly URL zkratku na původní URL. Omylem jsem zjistil, že existují dokumenty, které přesměrovávají samy na sebe -- příkladem budiž tento dokument. Z toho vyplývá, že bude třeba nějakým způsobem řešit zacyklení. Navrhuju řešení a lá UNIXové symlinky: sledovat jenom konečný počet přesměrování (je to v uvedené funkci).
Dále třeba takový http://www.youtube.com/cokoliv ústí ve 404 -- bude potřeba vykoumat, co všechno posílají prohlížeče, abychom se tvářili co nejvíce jako jeden z nich. Tohle se mi zatím nedaří. :-(
V současnosti mám

```
 7 OUR_FAKE_HEADERS = { "connection":"keep-alive",
 8     "cache-control":"max-age=0",
 9     "user-agent":"Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Fi
10     "accept":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.
11     "accept-encoding":"gzip,deflate,sdch",
12     "accept-language":"en-US,en;q=0.8",
13     "accept-charset":"ISO-8859-1;q=0.7,*;0.3"
14     }
```

a to (na youtube) nestačí, pořád vrací 404
Chtělo by asi zjistit, jestli nám takhle vzdoruje hodně webů, protože youtube asi není pro NLP kritický portál, že.

```
60 def final_url(url_text):
61     actual_url_text = url_text
62     max_redirs = 30
63     for i in range(1,max_redirs): # a do-while loop
64         actual_url = urlparse.urlparse(actual_url_text)
65
66         h = httplib.HTTPConnection(actual_url.netloc)
67
```

```
 68          params = urllib.urlencode({})
 69          headers = OUR_FAKE_HEADERS
 70          h.request("HEAD", actual_url.path, params, headers)
 71
 72          response = h.getresponse()
 73
 74          print actual_url_text, " - ", response.status
 75          if response.status not in [301,302,303]:
 76              return actual_url_text
 77
 78          headers = response.getheaders()
 79      #?    print headers
 80
 81          stamp = {}
 82          for header in headers:
 83              if header[0] == "location":
 84                  actual_url_text = header[1]
 85                  break
 86          else:
 87             return "Some kind of black magic was applied on the headers"
 88      else:
 89          return "Failed to find final destination (followed " + str(max_redirs) + "
redirections total)"
```

@Standa: Cookies... hmmm. Nevim, co bych tam měl řešit, naznač, cos měl na mysli. Spíš jsem měl na mysli, jestli některé weby si nenechají uložit cookies pro nějaký přístup na další stránky či co.. Asi je to nesmysl.. Ale nějaký případ typu: dám ti cookies, které ti pak poslouží jako klíč pro přístup sem a sem.. Ale asi je to nesmysl. No každopádně mám pocit, že jsem je do rrslib musel přidat, protože se někteér stránky bez cookiejaru chovaly dost divně.

2.2 Ukládání dat a jejich změn do úložiště

- jaký je rozdíl mezi daty pro page change monitoring a např. PDF soubory (články?)
- možnost uložit nebo neukládat obsah (např. když je příliš velký)
    - Definovat, co je velký soubor: nemůžeme nechat “uživatelský projekt”, aby tam ukládal 300MB soubory, protože si vývojář myslel, že je to dobrý nápad..

2.2.1 Relační databáze

- - sha1 nebo md5 pro checksum dat? Jakou roli bude hrát v identifikaci v db (jako PK?)
- asi PostgreSQL
- nebo kombinovat s NoSQL?

2.2.2 NoSQL databáze

- zvážit použití NoSQL databáze,
- jaký druh? (document store, případně column store)

pár odkazů pro informace o NoSQL:
- http://www.vineetgupta.com/2010/01/nosql-databases-part-1-landscape/
- http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html
- http://nosql-database.org/
- http://en.wikipedia.org/wiki/NoSQL

MongoDB
Výhody:
- nativní podpora dokumentů jako obsahu db (document store)

- versioning lze řešit jednoduše - mongo to má optimalizované pro tento účel a nebo ještě lépe lze soubory ukládat do GridFS, který má versioning nativně (ale vše replikuje!)
- ukládá automaticky i s MD5 a content-type souboru (hodně podobné s tím, co dává HTTP header)
- velmi rychlé a škálovatelné
- http://blog.pythonisito.com/2012/05/gridfs-mongodb-filesystem.html
- http://api.mongodb.org/python/current/api/gridfs/index.html
- http://techblog.floorplanner.com/post/20528556744/mongodb-and-gridfs-versioning
- http://www.abclinuxu.cz/clanky/programovani/lehky-uvod-do-mongodb

Nevýhody:

- GridFS nedokáže říct diff verzí.
- GridFS kládá celou novou verzi (ne diff jako git/svn) Ale samozřejmě lze to používat tak, že to budeme ukládat celé a pak si diff uděláme nějakým toolem (bsdiff).
- není relační, je třeba vymyslet, jak uchovávat vztah content-url 1:N při zachování možnosti URL jako indexu (jde to! mongo umí vyhledávat i podle klíče, který je v seznamu)

Instalace
ideální je asi z tgz, z balíku pro RHEL (10gen) neobsahuje mongofiles, což potřebujeme (GridFS)

Python
Pro MongoDB existuje pythoní knihovna pymongo pro přístup z Pythonu. http://pypi.python.org/pypi/pymongo/#downloads

Dál existuje ORM pro mongo: http://merciless.sourceforge.net/ ale to možná ani nebude potřeba.

Update
db.webarchive.update({"_id":ObjectId('4fe1ce22b4646d21ad000009')},
{"$addToSet":{"url": "http://www.google.com"}}, False)
Indexy
http://stackoverflow.com/questions/5912661/pymongo-mongodb-create-index-or-ensure-index

- zjistit co je ttl v ensure_index!
- rozdil mezi ensure index a create index

2.2.3 Ukládání změn - binární diffy

- - budeme potřebovat nějaký diff tool pro textové i **binární** soubory. Pro textové budeme používat asi diff, pro binární pak bsdiff/jdiff/xdelta. Otestovat hlavně na PDF, DOC, ODT souborech!!!
- - pro binární diffy se také podívat na velikost diffů, jestli není někdy výhodnější uložit celý nový soubor, než ukládat binární diff, který může nabobtnat

Testovací data (různé soubory):
PDF : 47 489 KB, 38 souborů
DOC : 9 583 KB, 20 souborů
ODT : 3 600 KB, 22 souborů
PS : 15 809 KB, 29 souborů
RTF : 5 990 KB, 21 souborů

Data jsem získal stylem : Google(“scientific article filetype:XXX”) => prvních 20+-2 výsledků. K tomu cca. 10 PDF a PS z Minervy a cca 10 bakalářek z UPGM.

Testovací data pro testování diffů verzí (podobné soubory):
Dvě sady -- jedna od S. Hellera (bakalářka), druhá od K. Beneše (100 odstavců Lorem Ipsum). Obě sady jsou ve dvou variantách: PDF a ODT. Lorem Ipsum je celkem v dvakrát osmi souborech, bakalářka v dvakrát pěti. Změny jsou v obou sadách umělé. Verze bakalářky mají úhrnem 1055KB (ODT), resp. 4048 KB (PDF), verze Lorem Ipsum pouze 151 KB (ODT), resp. 488 KB (PDF).

Postup testování:
Soubory každého druhu každý s každým (tj. každé pdf proti každému pdf), a.pdf x b.pdf i b.pdf x a.pdf, což by mělo rozumně srovnat odchylky.
Testováno na lokálu, tj. doma, tj, dneska jiz na low-end PC. (AMD Athlon 64 X2 Dual Core Processor 5200+ × 2, 4 GB RAM, Ubuntu 11.10, kernel 3.x.x)
Časy získány pomocí pythoního time.time(), zhruba takto:

```
t_start = time.time()
...processing (xdelta nebo jdiff...)
t = time.time() - t_start
```

Vysvětlení jednotlivých sloupců v tabulkách výsledků:

- **celkový čas** - součet těch časů z toho kódu výše
- **celková data** - souhrnná velikost všech vzniklých souborů - diffů
- **čas na 1 MB výsledku** - celkový čas/celková data, tj. poměr počítání na zapisování
- **čas na 1 diff** - celkový čas/počet vzniklých diffů, tj. průměrný (aritmeticky průměrný) čas na vytvoření jednoho diffu

Výsledky - nesouvisející dokumenty

xdelta:
xdelta delta soubor1 soubor2 vysledek

xdelta 1.1.3	Celkový čas	Celková data	Čas na 1 MB výsledku	Čas na 1 diff
PDF	667,149 s	1605 MB	0.415 s	0.462 s
DOC	61.305 s	112 MB	0.547 s	0.153 s
ODT	26.847 s	71 MB	0.378 s	0.055 s
PS	139.829 s	85 MB	1.645 s	0.166 s
RTF	39.607 s	19 MB	2.084 s	0.089 s

jdiff:
jdiff soubor1 soubor2 vysledek

jdiff 0.8 beta	Celkový čas	Celková data	Čas na 1 MB výsledku	Čas na 1 diff

PDF	2323.559 s	1711 MB	1.358 s	1.609 s
DOC	491.835 s	164 MB	3.000 s	1.229 s
ODT	272.921 s	73 MB	3.738 s	0.563 s
PS	1020.063 s	433 MB	2.355 s	1.212 s
RTF	447.961 s	109 MB	4.109 s	1.015 s

bsdiff:
bsdiff soubor1 soubor2 vysledek

bsdiff z r. 2003	Celkový čas	Celková data	Čas na 1 MB výsledku	Čas na 1 diff
PDF	11571.030 s	1609 MB	7.191 s	8.013 s
DOC	1016.501 s	114 MB	8.916 s	2.541 s
ODT	346.148 s	71 MB	4.875 s	0.715 s
PS	2532.740 s	72 MB	35.177 s	3.011 s
RTF	564.607 s	16 MB	35.287 s	1.280 s

Zajímavé údaje by byly:
- jak se chovají jednotlivé algoritmy v případě diffování souborů, které vycházejí z jednoho (tzn různé verze jednoho dokumentu, které mají nějakou timeline)

Poznatky, závěry:
- Je možné, že jdiff má problém se srovnáváním dost rozdílných dokumentů, protože je dost heuristický
- srovnání jak se chovají jednotlivé algoritmy v případě diffování obrázkových a textových dat (dokumenty obsahující hodně obrázku versus dokumenty obsahující samý text, ale formátovaný)j bude pravděpodobně k ničemu, protože většinou srovnáváme vědecké články, které jsou převážně textové.
- bsdiff je bezkonkurenčně nejpomalejší. Ve srovnání s xdelta produkuje zhruba stejně velké diffy, ale 10x až 20x pomaleji. Jdiff tvoří, mimo ODT, o HODNĚ větší diffy.

Výsledky - dokumenty s timeline

Pozn. “LI” = Lorem Ipsum, “BC” = bakalářka
xdelta:
xdelta delta soubor1 soubor2 vysledek

xdelta 1.1.3	Celkový čas	Celková data	Čas na 1 MB výsledku	Čas na 1 diff

LI - ODT	0.617 s	736 KB	0.838 s	0.009 s
LI - PDF	1.472 s	3000 KB	0.490 s	0.023 s
BC - ODT	0.805 s	1467 KB	0.548 s	0.032 s
BC - PDF	3.954 s	5451 KB	0.725 s	0.158 s

jdiff:
jdiff soubor1 soubor2 vysledek

jdiff 0.8 beta	Celkový čas	Celková data	Čas na 1 MB výsledku	Čas na 1 diff
LI - ODT	27.824 s	724 KB	38.430 s	0.434 s
LI - PDF	30.461 s	3009 KB	10.123 s	0.475 s
BC - ODT	14.386 s	1489 KB	9.661 s	0.575 s
BC - PDF	255.374 s	7074 KB	36.100 s	10.214 s

bsdiff:
bsdiff soubor1 soubor2 vysledek

bsdiff z r. 2003	Celkový čas	Celková data	Čas na 1 MB výsledku	Čas na 1 diff
LI - ODT	3.520 s	733 KB	4.802 s	0.055 s
LI - PDF	13.411 s	3007 KB	4.459 s	0.209 s
BC - ODT	12.087 s	1471 KB	8.216 s	0.483 s
BC - PDF	65.563 s	5294 KB	12.384 s	2.622 s

S ohledem na menší rozměr testovacích dat jsem provedl měření dvakrát. Odchylka byla v řádu max. jednotek procent, tj. považuju je za spolehlivé. (vzhledem k daným datům)

Poznatky, závěry:
- Nasazením dokumentů s timeline se sice obrátila situace bsdiff - jdiff, ale xdelta je pořád řádově rychlejší, při přibližně stejné velikosti vzniklých diffů. Případné rozdíly ve velikosti (xdelta - bsdiff) vzniklých dat nepřesahují 15 %.

2.3 md5 checksum

Testovací data (různé soubory, předpony dle SI):
Pětkrát soubor o velikosti 1 KB, 100 KB, 1 MB, 5 MB a 10 MB (tj. celkem 25 souborů).
Vygenerovány pomocí `openssl rand velikost > soubor`.

@Karel: mohl bys prosím ještě stejným způsobem otestovat několik PDF souborů (třeba tu bakalářku, to je fuk), jestli to dává aspoň rámcově stejné časy (tzn jestli to nedělá nějaké šaškárny nad binárními soubory, ale nemělo by). Nemusíš dělat takhle hlubokou analýzu a srovnání, jen tam hodit fakt pár souborů o nějakých rozumných velikostech, co se blíží těmto.. Chtěl bych se jen ujistit, že je to MD5 opravdu tak rychlé.

Ok, pustil jsem to (na lokále), jsou tam dva 11 MB > DOC > 8 MB, dvě 9 MB > PDF > 7 MB, dva tvoje největší bakalářka-soubory (ODT i PDF), 10 mega itercí na soubor. Pustil jsem to v testu i s těmi umělými daty, ať je to trochu vypovídající: Čas na jeden md5checksum **je v rozmezí 2.85 us - 3.52 us**, při zanedbání nejvyšší hodnoty (bp_xhelle_04.odt) a nejnižší (1KB umělý soubor) dokonce **v rozmenzí 2.96 us - 3.20 us**. Takže v tom valný rozdíl není.

Metodika testů:
Na Atheně 2. Deset milionů iterací na každý soubor.

Výkonný kód pro pythoní řešení:

```
mdfiver = hashlib.md5()
mdfiver.update(str(os.read(fd,10000000)))
for i in range (1,n_iters):
        mdfiver.digest()
t_end = time.time()
```

hashlib	Soubor 1	Soubor 2	Soubor 3	Soubor 4	Soubor 5
1 KB	11.766 s	11.729 s	11.638 s	11.679 s	11.638 s
100 KB	12.123 s	12.013 s	12.191 s	12.104 s	12.124 s
1 MB	11.966 s	11.966 s	11.993 s	11.950 s	11.952 s
5 MB	11.982 s	11.997 s	12.047 s	12.075 s	12.041 s
10 MB	12.070 s	12.127 s	12.097 s	12.042 s	12.346 s

Poznatky a závěry:
Prvotní pythoní testy (hashlib) na lokále ukazují, že na velikosti vůbec nezáleží.

Hypotéza z lokálu se potvrdila, hashlib.md5 je časově prakticky invariantí vůči velikosti souboru (v rozsahu 1KB - 10MB). Na atheně2 trvá výpočet jednoho md5checksumu tímto způsobem cca. 1,2us.

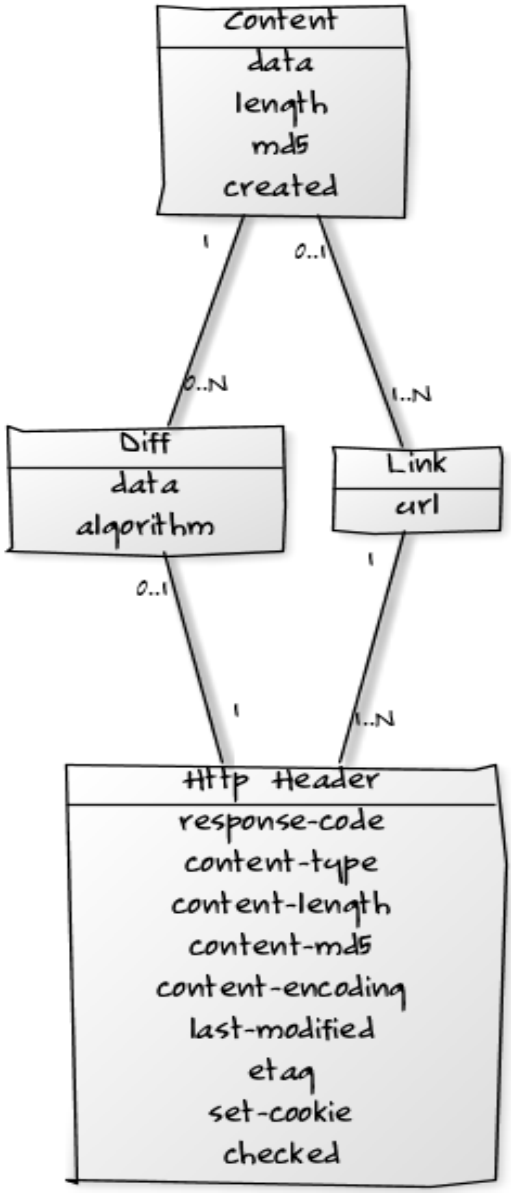
To je IMHO (Karel) čas “zanedbatelný” -- jsme schopni spočítat cca. 800k takových checksumů za vteřinu. @Karel: skvěle, toto je asi nejdůležitější poznatek. Díky.

3. Návrh - obecné, ERD, databáze, UML, systém,

ERD - datové modelování

Z ERD bude vycházet databázové schéma (pokud použijeme klasickou relační databázi). V případě použití NoSQL databáze se ale ERD bude hodit alespoň jako náznak toho, co bychom měli uchovávat za data. Bude to nutné samozřejmě denormalizovat.

Online kreslič UML a ERD:
http://yuml.me/diagram/scruffy/class/draw
http://yuml.me/diagram/scruffy/class/samples
Prozatimní verze ERD: http://yuml.me/32e1c193



Obecné

- **Velké soubory** - za velký soubor bude považován soubor s velikostí > 10MB. Takový soubor nebude vždy stahován, ale pouze se bude kontrolovat, jestli se nezměnila HTTP hlavička kterou server odešle.
- **Typ dotazů** - Obecně bude nejvíce dotazů typu **HEAD - opravdu???**
- **Url shortening -** Bude třeba rozdělit pohled na url a obsah pod ní, vztah bude tedy 1:N. Obsah bude považován za stejný, pokud má stejný md5 hash.

Databáze

- do db ukládat metadata o souboru/dokumentu (content-type, content-length atp. => celou HTTP hlavičku??)
- o db se bude ukládat pouze aktuální verze dokumentu a pak zpětné diffy - co diff, to HTTP header, ale když je header 404 nebo 304, tak tam samozřejmě diff není!
- HTTP headery se budou ukládat pouze pokud se v nich něco změní nebo pokud se změní obsah
- počítání MD5 (v Postgre to nějak nešlo, v Mongo je to automaticky)
- název schématu: **webarchive**
- - a další věcí (TODO: DOPLNIT)

Kód pro náš ER diagram (nedokončené):

```
[Content|data;length;md5;created]0..1-1..N[Link|url],
[Link]1-1..N[HttpHeader|response-code;content-type;content-length;content-
md5;content-encoding;last-modified;etag;set-cookie;checked],
```

```
[Content]1-0..N[Diff|data;algorithm],
[Diff]0..1-1[Http Header]
```

Návrh NoSQL databáze (schema-free document-oriented)

Založeno na MongoDB.

```
db=webarchive
collections=[db.content (GridFS), db.httpheader]

content = {
      _id:              ObjectId('905464b98b63a213c708d00'),
      filename:         "index.html",
      upload_date:      "18.3.2012 18:32:24",
      encoding:         "utf-8",
      content_type:     "text/html",
      md5:              "automaticky generovany hash",
      sha1:             "vypocitany sha1 hash obsahu",
      length:           84321, (melo by se shodovat s HTTP content-length)
      urls:             ["http://www.google.com","http://www.bit.ly/h8sf3a1"],
      data:             "<html>...",
}

httpheader = {
      _id:              ObjectId('68747470733a2f2f7777772e'),
      url + uniq index: "http://www.google.com",
      timestamp:        cas prijmuti headeru
      response_code:    200,
      last_modified:    "Wed, 20 Jun 2012 12:55:22 GMT",
      content_length:   175943,
      etag:             "Ld7XRovMP3d8aXcWaLIge3x",
      uid:              "rrs_deliverables",
      content:          ObjectId() // pokud byl response code<400
}
```

Knihovna rrslib

UML

Kód pro UML knihovního backendu (nedokončené):

```
[Monitor|string user_id;string db_host;int db_port;string db_name|
get(string url);allow_large_documents();check_uid();check_multi(list
urls)]-[MonitoredResource|string url;string uid;Storage storage;Resolver
resolver;File file|
check();get_last_version();get_version(time_or_version);get_diff(start,
end);available(httptime);last_checked()]
[MonitoredResource]-[File|cosi()]
[File]-[Content|data;md5;created|+get_revision_at(datetime)]
[BaseMongoModel]<-[Storage]
```

Poznámky:

- Název knihovního modulu: **changemonitor**
- Umístění: rrslib.web.changemonitor