UNIVERSITÀ DEGLI STUDI DI PADOVA
FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

CRAWLING DEL WEB ITALIANO

VALUTAZIONE DELLE SOLUZIONI E GESTIONE DEL CRAWLING

GIANLUCA PENGO

RELATORE:
PROF. LUCA PRETTO
CORRELATORE:
PROF. ENOCH PESERICO

ANNO ACCADEMICO 2009/2010

30 SETTEMBRE 2010

UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

Sommario

Gli algoritmi di link analysis tentano di dedurre la qualità di un documento dalla struttura topologica del grafo che rappresenta il web nella sua interezza o in un suo sottoinsieme. Presso l'Università di Padova si sta svolgendo un progetto di ricerca che tra i suoi obiettivi ha quello di valutare l'efficacia degli algoritmi di link analysis, rispetto alla determinazione della qualità dei documenti. Questa relazione tratta lo studio di fattibilità e la realizzazione di un sistema di crawling per l'ottenimento del grafo del web italiano, che sarà utilizzato poi nel progetto di ricerca. All'interno dello studio di fattibilità sono analizzati e valutati alcuni crawler open source ed è motivata la scelta di utilizzare il crawler Heritrix, tra quelli presentati. Vengono esposte le diverse soluzioni proposte per la creazione del modulo aggiuntivo, che s'integra ad Heritrix, per l'ottenimento del grafo del web. Sono poi trattate le strutture dati utilizzate per la memorizzazione e la loro finalizzazione nella fase di post-crawling, sono riportati i risultati dei test effettuati nelle diverse configurazioni del sistema, utilizzati per il dimensionamento e la configurazione del sistema nel suo complesso. Infine è esposto l'utilizzo di un limitatore di banda, che modifica automaticamente il limite nelle diverse ore del giorno, e un controllore delle connessioni contemporaneamente attive integrato al crawler, per sopperire ai limiti di banda e di connessioni TCP contemporanee imposti dalla rete della struttura in cui si svolge il crawling. Questo lavoro ha portato all'implementazione di un sistema di crawling adatto ad operare in condizioni di limitate risorse hardware e di rete. L'obiettivo perseguito nello sviluppo di questo lavoro è l'esecuzione di un crawling, il più possibile esaustivo del web italiano, nel minor tempo possibile e limitando le risorse necessarie, al fine di ottenere un grafo che rappresenti la struttura topologica del web italiano.

Indice

1. Introduzione

Il web oggi è composto di miliardi da documenti, contenenti informazioni con caratteristiche molto eterogenee. Per ricercare le informazioni d'interesse è necessario utilizzare i motori di ricerca. I motori di ricerca inizialmente erano impiegati su collezioni di documenti su cui operavano sistemi di Information Retrieval[1] tradizionali. L'Information Retrieval tradizionale opera su collezioni di documenti di qualità omogenea. I motori di ricerca, di conseguenza, basavano i loro algoritmi sulla rilevanza dei documenti rispetto a un'interrogazione dell'utente. Con l'avvento del web questi motori di ricerca sono stati adattati per essere utilizzati in questo nuovo scenario, dovendo però modificare i loro algoritmi. Nel web infatti chiunque può pubblicare qualsiasi tipo d'informazione. Il web può essere considerato quindi come una collezione di documenti eterogenei, con caratteristiche d'importanza, autorevolezza, qualità molto diverse. Non è quindi possibile valutare un documento considerando solo la sua rilevanza, ma è necessario stimarne la qualità utilizzando per esempio algoritmi di link analysis (tra i primi proposti per questo scopo). Questi algoritmi deducono la qualità di un documento dalla struttura topologica di un grafo orientato, che rappresenta tutto il web o un suo sottoinsieme.

Presso l'Università di Padova si sta svolgendo un progetto di ricerca che tra i suoi scopi ha quello di valutare l'efficacia dei principali algoritmi di link analysis, rispetto la determinazione della qualità dei documenti. Per tale scopo il progetto mette a confronto i punteggi che algoritmi di link analysis, come Page Rank [27] e HITS [20], assegnano ai documenti, con i giudizi di qualità espressi da utenti umani, per dedurre in quale misura tali algoritmi esprimono la qualità effettiva dei documenti. Al progetto hanno per ora partecipato due gruppi di studenti. Un gruppo si è occupato di vagliare le possibili soluzioni per ottenere un grafo del web, che rappresenti il più fedelmente possibile il web italiano, e di implementare la soluzione scelta. Il secondo gruppo si è occupato della scelta e implementazione di una tecnica di campionamento delle pagine restituite dai motori di ricerca, in risposta ad alcune interrogazioni specifiche, e alla realizzazione di un sito web per la raccolta delle valutazioni di qualità delle pagine (ottenute

[1] Parte della computer science: studia il recupero dell'informazione (non dei dati) da una collezione di documenti scritti. I documenti recuperati devono soddisfare l'esigenza informativa dell'utente, solitamente espressa in linguaggio naturale[6].

con il precedente campionamento), espresse da una popolazione di utenti volontari che contribuiranno al progetto. Per ottenere il grafo del web si effettuerà un crawling il più possibile esaustivo del web italiano. Il web italiano è stato scelto come dominio in cui effettuare la ricerca per diversi motivi. In primo luogo, le dimensioni dell'intero web rendono in sostanza impossibile effettuarne il crawling (in tempi ragionevoli e con risorse limitate) ed eseguire la successiva valutazione. In secondo luogo, per ragioni di lingua il web italiano è poco puntato dall'esterno e punta poco verso l'esterno, questo vuol dire che pagine web non appartenenti al dominio ".it" raramente contengono link verso pagine del dominio .it e viceversa. Questo permette di considerare il web italiano come un sottografo isolato dell'intero web mondiale, quindi è ragionevole ipotizzare, per la caratteristica di frattalità del web [15], che il dominio in questione abbia una struttura analoga a quella dell'intero web, in termini di valori di rilevanza, PageRank, indegree medio e altri parametri. Infine, essendo italiano il gruppo di utenti che valuterà le pagine, la scelta del dominio .it agevola e rende più efficace la loro valutazione.

In questa relazione sarà esposto, dopo una breve introduzione all'argomento, lo studio di fattibilità per l'implementazione del crawler e la sua successiva realizzazione e gestione, perseguendo l'obiettivo di ottenere il grafo del web italiano nel minor tempo possibile e con il minimo impiego di risorse necessarie.

2. Crawling

Il *web crawling* è un prerequisito allo studio e analisi della struttura del web. Consiste in un processo nel quale si collezionano documenti reperibili nel web. L'obiettivo del crawling è raccogliere efficientemente e velocemente più documenti "utili" possibili, assieme alla struttura dei collegamenti ipertestuali che li interconnettono.

In questo primo capitolo saranno fornite le nozioni basilari sul crawling, necessarie ad affrontare problematiche e soluzioni, esposte nei successivi capitoli.

2.1. Crawler

Un crawler (detto anche spider o robot), è un software che analizza i contenuti di una rete o di un database in maniera sistematica e automatizzata. Esporremo ora in breve, le operazioni eseguite da ogni crawler basato su collegamenti ipertestuali nel web, in un'intranet o altre collezioni di documenti ipertestuali. Il crawling ha inizio con uno o più URL che costituiscono un insieme di semi (seed). Seleziona un elemento da questo insieme e scarica il documento da quell'URL, il quale è poi analizzato, per estrarre sia il testo, sia i collegamenti ipertestuali (ognuno dei quali punta a un altro URL). Il testo estratto alimenta un indicizzatore di testo, mentre i collegamenti sono aggiunti nell'Url frontier (frontiera), il quale consiste negli URL corrispondenti alle pagine che non sono ancora state processate dal crawler. Inizialmente la frontiera contiene l'insieme dei seed. Quando una pagina è scaricata, il corrispondente URL è eliminato dalla frontiera. L'intero processo può essere visto come l'attraversamento del grafo del web. Nel crawling continuo, l'URL delle pagine scaricate è aggiunto in fondo alla frontiera, per essere nuovamente scaricato in futuro. L'implementazione di questo apparentemente semplice attraversamento del grafo del web è complicata in crawler reali, dove sono richieste alcune caratteristiche necessarie, come robustezza e politeness, e altre che comunque dovrebbero essere presenti, come l'essere distribuito, scalabile, estensibile, di qualità, flessibile, efficiente, ad alte prestazioni e garantire la "freschezza" dei documenti.

2.1.1. Caratteristiche

Analizziamo ora brevemente queste caratteristiche:

Robustezza: il web contiene server che creano spider traps, generatori di pagine web che ingannano il crawler portandolo a scaricare un infinito numero di pagine in un particolare dominio. I crawler devono essere progettati per essere resistenti a queste trappole. Non tutte le trappole sono maliziose, alcune sono effetti indesiderati di un cattivo sviluppo di un sito web. Inoltre deve presentare un alto grado di tolleranza, nei confronti dei malfunzionamenti hardware e software.

Politeness: i web server hanno sia esplicite che implicite politiche che regolano la frequenza con cui un crawler può visitarli. Queste politiche devono essere rispettate.

Distribuito: il crawler può avere la possibilità di essere eseguito in maniera distribuita su più macchine.

Scalabile: l'architettura del crawler dovrebbe permettere l'aumentare della velocità di crawling, aggiungendo macchine e banda aggiuntive.

Prestazioni ed efficienza: il sistema di crawling dovrebbe fare un uso efficiente delle varie risorse di sistema, incluso il processore, lo storage e la banda di rete.

Qualità: poiché, numerosi documenti reperibili nel web, sono di scarsa utilità per l'obiettivo del crawling, il crawler dovrebbe propendere verso lo scaricare documenti "utili".

Freschezza: in molte applicazioni, il crawler deve operare in modo continuo: per ottenere le copie aggiornate delle pagine precedentemente scaricate. Il crawler di un motore di ricerca, per esempio, dovrebbe assicurare che gli indici del motore di ricerca contengono una rappresentazione abbastanza aggiornata di ogni pagina indicizzata.

Estensibile: il crawler dovrebbe essere progettato per essere estensibile, per esempio per far fronte a nuovi formati dei dati, nuovi protocolli di scaricamento, e così via. Questo richiede che l'architettura del crawler sia modulare.

Flessibilità: capacità del crawler di adattarsi a diversi ambiti applicativi, attuando il minor numero di modifiche possibili.

2.1.2. Politiche di crawling

Esistono differenti politiche di crawling tipicamente utilizzate nei diversi ambiti di applicazione per l'acquisizione di dati. Di seguito sono riportate le principali politiche di crawling con cenni agli ambiti in cui vengono utilizzate.

Breadth-first crawling: Utilizzate per costruire grandi motori di ricerca o repository[1]. Utilizzano crawler a elevate prestazioni, che partono da un piccolo insieme di pagine ed esplorano le altre seguendo i collegamenti ipertestuali, applicando la visita in larghezza (in inglese breadth-first). Nella realtà, molto spesso le pagine non sono visitate applicando uno stretto "breadth-first", ma usando una varietà di politiche, per esempio privilegiando il completamento del crawling di un sito web o scansionando prima le pagine più importanti[2].

Crawling continuo: Una volta che i documenti sono stati inizialmente acquisiti, sono periodicamente scaricati e controllati, per aggiornare gli indici in caso di modifiche. Nel caso più semplice, questo può essere fatto eseguendo un altro crawling breadth-first, o semplicemente richiedendo nuovamente tutti gli URL nella collezione. Tuttavia, una varietà di euristiche possono essere impiegate per eseguire più frequentemente il recrawling dei documenti, dei siti, o i domini più importanti. Una buona strategia di recrawling è cruciale per mantenere aggiornati gli indici di ricerca con una banda limitata. Recenti lavori condotti da Cho and Garcia-Molina [11, 12] hanno mostrato tecniche per ottimizzare la "freschezza" delle collezioni data l'osservazione della cronologia degli aggiornamenti.

Focused crawling: Motori di ricerca più specializzati possono utilizzare politiche che tentano di focalizzarsi in uno specifico argomento, in un particolare linguaggio o per esempio nelle immagini. In un approccio più generale sono state proposte analisi sulla struttura dei link [10, 29] e tecniche di apprendimento automantico[3] [14, 24]. L'obiettivo del crawling focalizzato è trovare il maggior numero di pagine d'interesse senza usare troppa banda.

Passeggiata aleatoria e campionamento: Sono state studiate molte tecniche che usano la passeggiata aleatoria (random walks) nel grafo del web per campionare pagine o stimare la dimensione e la qualità dei motori di ricerca [30, 19, 18].

Crawling "hidden web": Molti dati accessibili via web attualmente risiedono in basi di dati e possono essere recuperati solo ponendo le appropriate interrogazioni e/o compilando form nelle pagine web. Recentemente, molto interesse è stato concentrato nell'accesso automatico a questi dati, chiamati "Hidden Web", "DeepWeb", or "Federated Facts and Figures". Un crawler come quelli qui de-

[1] Un repository (che può essere tradotto con il termine deposito) è un ambiente di un sistema informativo, in cui vengono gestiti i metadati, attraverso tabelle relazionali.

[2] Vedi [13] euristiche che tentano di effettuare prima il crawling delle pagine più importanti e [22] risultati sperimentali dimostrano che anche uno stretto breadth first troverà velocemente pagine con alto PageRank.

[3] L'apprendimento automatico (noto in letteratura come Machine Learning) - una delle aree fondamentali dell'Intelligenza Artificiale - si occupa della ricerca di metodi algoritmici per sviluppare programmi che automaticamente, basandosi su dati empirici, migliorano la propria performance nel tempo.

scritti, può essere esteso e usato come un efficiente interfaccia per questo crawling [23]. Tuttavia, ci sono molte sfide associate all'accesso all'hidden web e l'efficienza dell'interfaccia non è più il problema principale.

Notare che ci sono significanti differenze tra le diverse politiche. Per esempio un crawler breadth-first deve tenere traccia di quali pagine sono già state scansionate: questo è generalmente realizzato utilizzando una struttura dati "URL visitati" che potrebbe risiedere su disco nei crawl di grandi dimensioni. Un crawler focalizzato sull'analisi dei collegamenti, potrebbe invece utilizzare strutture dati addizionali per rappresentare la struttura del grafo della parte del web scansionato e un classificatore per giudicare la rilevanza di una pagina [29, 10], ma la dimensione della struttura potrebbe essere molto più piccola. Ci sono però, numerose azioni che richiedono di essere svolte nella maggior dei casi, come l'applicazione dei robot exclusion, la limitazione della velocità di crawling, la risoluzione dei DNS e altre ancora. Questa serie di azioni potrebbe essere svolta da un nucleo essenziale del software, il quale può essere impiegato come base per lo sviluppo di crawler specifici per ogni ambito di applicazione.

2.1.3. Architettura di un crawler

Una semplice architettura di un crawler potrebbe essere composta di diversi moduli che collaborano tra di loro, come mostrato in Figura 2.1 (tratta da [21]).

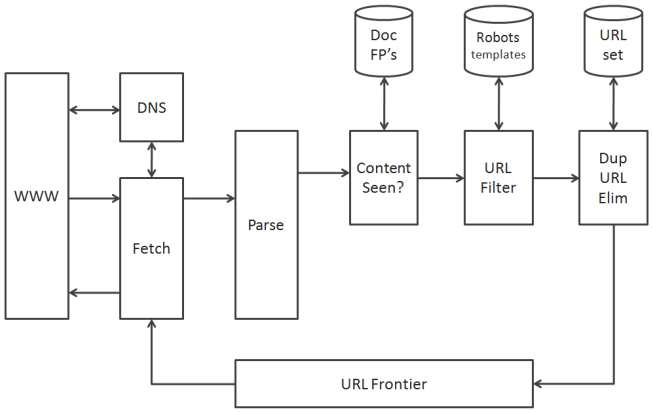

Figura 2.1.: Architettura base di un crawler.

- URL frontier contiene gli URL che devono essere scaricati nel crawling corrente.
- Il modulo di risoluzione dei DNS determina il server web da cui scaricare la pagina specificata dall'URL.

- Il modulo per il download dei documenti utilizza il protocollo http per reperire la pagina all'URL indicato.

- Il modulo per l'eliminazione dei duplicati determina quando un link estratto è già nella frontiera degli URL o è stato recentemente scaricato.

In crawler ad alte prestazioni, in cui diverse centinaia o migliaia di documenti devono essere processati al secondo, i componenti sono replicati per elevare le prestazioni, così da superare i diversi colli di bottiglia del sistema, al fine di ottenere elevate velocità di elaborazione dei documenti. Il crawling, quindi, può essere eseguito da uno qualunque dei centinaia di potenziali thread, ognuno dei quali esegue il ciclo sopra illustrato. Questi thread possono essere eseguiti da un singolo processo, o essere partizionati su più processi in esecuzione su nodi diversi di un sistema distribuito. Iniziamo assumendo che la frontiera di URL risieda nella stessa macchina in cui è eseguito il thread e non sia vuota. Un thread inizia richiedendo un URL alla frontiera e scaricando il documento reperibile a quell'URL, generalmente usando il protocollo http. Il documento scaricato è memorizzato in un "deposito" temporaneo, dove vi sono eseguite alcune operazioni. Successivamente la pagina è analizzata e il testo e i link al suo interno sono estratti. Il testo (con tutti tag d'informazione) sono passati all'indicizzatore. Le informazioni sul link, inclusa l'anchor text, sono anch'esse passate all'indicizzatore per essere impiegate nel ranking utilizzato poi dal motore di ricerca. Inoltre, ogni link estratto attraversa una serie di test per determinare se può essere inserito nella frontiera degli URL. Nel primo controllo, il thread verifica se una pagina con lo stesso contenuto è già stata visitata su un altro URL. Il modo più semplice per implementarlo è usare un'impronta digitale (fingeprint), come per esempio il checksum (posto poi in un archivio chiamato "Doc FP's").

Successivamente, un filtro URL, basato su diversi controlli, è utilizzato per determinare se l'URL estratto, potrebbe essere escluso dalla frontiera. Per esempio, il crawling potrebbe decidere di escludere alcuni domini o accettarne solo alcuni (per esempio solo URL .it).

Molti host nel web, mettono alcune porzioni del loro sito web non raggiungibili ai crawler, grazie a uno standard che prende il nome di *Robots Exclusion Protocol*. Questo è fatto mettendo un file con il nome di *robots.txt* nella cartella principale del sito. Il file potrebbe per esempio specificare, che la cartella dei file temporanei non deve essere scansionata, ad eccezione del motore di ricerca interno al sito. Il file robots.txt deve essere quindi scaricato dal sito, al fine di verificare se l'URL in esame, passa le restrizioni dei robots e può quindi essere aggiunto alla frontiera. Anziché scaricare il file dei robots ogni volta, per testare ogni URL aggiunto alla frontiera, una cache può essere utilizzata per ottenerne la copia recente di un host. Questo è molto importante quando molti link estratti da una pagina rientrano

nell'host in cui la pagina è stata scaricata e più tardi saranno controllati per il robots.txt di quell'host. Pertanto, eseguendo il controllo durante il processo di estrazione del link, avremmo una località particolarmente elevata nel flusso di host che dobbiamo controllare per i file robots.txt, il che comporta un'alta frequenza di hit cache. Sfortunatamente, un URL (in particolare con bassa qualità o che cambia raramente) può restare nella frontiera per giorni o settimane, se eseguiamo il controllo sui robots prima di aggiungere l'URL alla frontiera, quando sarà scaricato il suo file robots.txt potrebbe essere cambiato. Dobbiamo quindi eseguire la verifica prima di scaricare la pagina. Nonostante tutto, mantenere una cache di file robots.txt è ancora altamente efficiente, infatti, c'è sufficiente località anche nel flusso di URL uscita dalla frontiera.

Successivamente, un URL potrebbe essere normalizzato. Spesso nel codice HTML, un link in un documento d indica il link verso un altro documento utilizzando un indirizzo relativo dal documento d, deve essere quindi trasformato in un indirizzo assoluto. Infine, l'ultimo controllo permette di eliminare i duplicati: se l'URL è già nella frontiera o è già stato scansionato, non sarà aggiunto alla frontiera. Quando un URL è aggiunto alla frontiera, gli è assegnata una priorità, in base alla quale è rimosso da essa per essere scaricato. Alcune operazioni di manutenzione sono di norma eseguite da un thread dedicato. Questo thread è generalmente inattivo eccetto quando è attivato per eseguire il log delle statistiche di crawling (URL visitati, dimensione della frontiera, ecc), decidere quando terminare il crawling o farne il checkpoint. Quando crea il checkpoint, un'immagine dello stato del crawler (nella maggior parte di casi della frontiera) è salvata su disco. Se il crawler dovesse interrompersi per un errore o per un guasto, il crawling ripartirebbe dall'ultimo checkpoint.

3. Studio di Fattibilità

In questo capitolo è riportato lo studio di fattibilità redatto per la scelta del sistema di crawling da realizzare.

3.1. Obiettivi dello studio

In questa prima sezione sono descritti gli obiettivi che si vogliono perseguire con la redazione di questo studio di fattibilità.

3.1.1. Descrizione degli obiettivi

Presso l'università di Padova si sta svolgendo un progetto di ricerca volto a valutare l'efficacia dei principali algoritmi di link analysis rispetto alla determinazione della qualità dei documenti, per la restituzione dei risultati in seguito alle interrogazioni degli utenti. Il progetto necessita di disporre del grafo del web italiano, dominio nel quale sarà condotta la ricerca, per poter calcolare i valori che algoritmi come Page Rank e HITS, assegnano ai documenti reperibili nel web, i nodi del grafo, e confrontare tali valori con quelli ottenuti utilizzando le valutazioni raccolte dagli utenti.

L'obiettivo di questo studio è valutare le possibili soluzioni d'implementazione di un sistema di crawling per l'ottenimento del grafo del web italiano, nel minor tempo possibile e con il minor dispendio di risorse.

3.1.2. Riferimento al piano di sviluppo del sistema informativo

Parallelamente a questo progetto, per il reperimento del grafo, ve n'è uno che mira al campionamento dei risultati restituiti dai principali motori di ricerca, in risposta ad alcune query specifiche, e alla realizzazione del sistema atto alla valutazione online, da parte degli utenti, delle pagine restituite dai motori di ricerca.

3.2. Analisi delle esigenze

Sono ora descritte le esigenze funzionali riscontrate e i metodi impegati per raccoglierle.

3.2.1. Descrizione dei metodi utilizzati

L'analisi delle esigenze che segue è stata ottenuta conducendo alcuni colloqui con i responsabili del progetto di ricerca e utilizzando materiale scientifico e non, riguardante lo svolgimento di crawling di domini molto estesi (maggiori di 100 milioni di pagine).

3.2.2. Descrizione delle esigenze funzionali

Dai diversi articoli disponibili in letteratura abbiamo riscontrato le seguenti caratteristiche, che un crawler dovrebbe presentare per processare domini molto estesi:

- Scalabilità
- Flessibilità
- Fault tolerance
- Politeness
- Essere distribuito
- Essere facilmente configurabile

Il crawler dovrà analizzare diverse tipologie di documenti presenti nel web quali: PDF, DOC, SWF; quindi non si limiterà alle sole pagine HTML.

3.2.3. Descrizione delle esigenze dei dati

Come esposto in precedenza, l'obiettivo di questo progetto consiste nell'ottenere il grafo di uno snapshot del web italiano (solo domini .it).

Il grafo è composto di nodi e archi orientati, i primi rappresentano i documenti presenti nel web, i secondi, i collegamenti tra di essi.

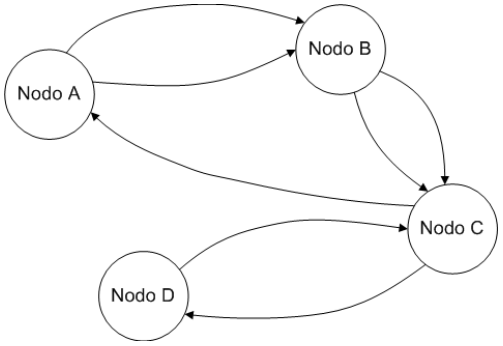

Figura 3.1.: Esempio grafo del web

Come si può notare, dall'esempio in Figura 3.1, gli archi sono orientati, dal documento/nodo in cui è presente il link a quello puntato da tale collegamento. Si rende quindi necessario, memorizzare solamente gli archi uscenti e non quelli entranti in un nodo. Per questo progetto non sono state fatte particolari richieste sulle modalità di memorizzazione né sulle successive modalità di accesso a tali dati.

3.3. Situazione di partenza

In questa sezione è descritta la tecnologia attualmente disponibile per il progetto, sono analizzati i prodotti reperibili sul mercato, che offrono una soluzione parziale o completa alle esigenze rilevate, e infine esposti i vincoli riscontrati.

3.3.1. Tecnologia disponibile

La macchina, o le macchine che saranno utilizzate per il crawling, saranno ospitate nel laboratorio di "ACG" del Dipartimento di Ingegneria dell'Informazione dell'Università di Padova. La banda a disposizione del laboratorio è di circa 40Mbit/s, condivisa con tutte le utenze dell'infrastruttura, la quale per la maggior parte del giorno ha un carico molto elevato ed è satura in alcune ore; durante la notte il carico è notevolmente minore, vengono però eseguite operazioni di back-up le quali occupano circa il 30-40% della banda, per alcune ore. Non è stato fatto alcun riferimento alle caratteristiche delle macchine che saranno messe a disposizione per il crawling, le quali saranno dimensionate in seguito durante il progetto.

3.3.2. Analisi di mercato

Viste le considerazioni esposte in precedenza, le principali soluzioni attuabili sono: lo sviluppo di un applicativo ad hoc o l'utilizzo ed eventuale adattamento di un prodotto open source disponibile in rete. Analizzando le disponibilità del mercato sono state trovate diverse soluzioni open source, le più rilevanti e adatte al progetto sono riportate di seguito:

Methanol

Methanol è un web crawler open source completamente configurabile. Permette di definire proprie regole sui tipi di file che devono essere inclusi nel crawling, supporta lo scripted filetype parsing, permette una grande varietà di personalizzazioni e può essere configurato per qualsiasi esigenza. Di seguito le caratteristiche principali:

- Progettato con l'obiettivo di ottimizzare la velocità
- È distribuito, parti diverse del sistema possono essere eseguite su server diversi
- Multi-threaded
- Sistema di estensione tramite moduli, supporto per l'uso di data parser, filtri, gestori di protocollo e tipi di documenti
- Download automatizzato
- Filtri per tipo di file definibili dall'utente (secondo tipi MIME, estensione del file o espressioni UMEX)
- Semplice ma potente filtraggio degli URL attraverso UMEX
- Altamente configurabile tramite linea di comando
- Supporto per gestione automatica dei cookie quando si esegue su http
- Robots Exclusion Standard
- Affidabile, rete fault-tolerant, rilevamento redirect-loop e di alcune spider trap
- Conversione da HTML a XML/XHTML
- Supporto MySQL attraverso JavaScript-MySQL
- Interfaccia Unix-friendly, piping in e out dei dati per il parsing e il crawling

- Portabile, testato su 32/64-bit Linux 2.6, 32/64-bit FreeBSD 6.x/7.0 e Mac OS X. Dovrebbe funzionare nella maggior parte degli OS Unix-like, parziale supporto per Windows

Methanol considera un crawler un insieme di regole, che definiscono come pagine web e file vengono scansionati. Il processo con i thread che scansionano le pagine web e i file, può essere considerato come un client con dei thread *"lavoratori"*. I *lavoratori* possono lavorare indipendentemente su pagine diverse, ma se eseguiti sotto lo stesso processo condividono la stessa URL cache e lo stesso insieme di regole. Un *lavoratore* è sempre connesso con un solo crawler alla volta, tuttavia a seconda della configurazione, i *lavoratori* posso dinamicamente cambiare crawler. La principale differenza tra due crawler è la loro lista di filetype: un crawler potrebbe per esempio avere una definizione per il filetype *html* e un altro per i file video. I *lavoratori* sono limitati ai tipi di file del loro crawler, un tipo di file a sua volta può avere un parser e degli attributi. Un parser è un piccolo script o una funzione che imposta degli attributi. Per esempio, potremmo avere il tipo di file *HTML* con gli attributi *title* e *description*. Il lavoro del parser è quello di estrarre i dati e impostare gli attributi. Quando un client è connesso a un sistema Methanol, i dati possono essere caricati sul sistema solo se gli attributi sono stati impostati dal parser e una volta che sono stati caricati sul sistema, i dati saranno disponibili nel database MySQL.

Methanol è giunto alla versione 1.7.0 (rilasciata nel giugno 2009), l'ultima versione può essere scaricata dal sito del progetto [1].

Heritrix

Heritrix Web Crawler è progettato per essere modulare. I moduli possono essere caricati quando esso è in esecuzione attraverso l'user interface. In questo modo gli utenti possono modificare o aggiungere moduli secondo le loro esigenze. Il crawler è interamente sviluppato in Java, è composto da un nucleo di classi e dai moduli aggiuntivi. Le classi del nucleo possono essere configurate ma non sostituite, mentre classi aggiuntive possono essere sostituite.

Segue una veloce panoramica delle classi principali:

Il CrawlController Comprende tutte le classi che devo cooperare per eseguire il crawling, fornisce un'interfaccia ad alto livello per eseguire il crawling ed esegue il "master thread" che distribuisce gli URL dal Frontier ai ToeThreads. I sottocomponenti comunicano tra di loro attraverso il CrawlController.

Il Frontier È il responsabile della distribuzione degli URL che devono essere scansionati e delle politiche di visita, che fanno in modo di non scansionare troppo pesantemente un server web. Una volta che un'URL è stato scansionato, esso è portato in fondo alla frontiera. Il Frontier mantiene inoltre lo stato del crawling, tenendo traccia di:

- quali URL sono stati scoperti

- quali URL devono essere processati
- quali URL sono già stati processati

ToeThreads Heritrix è multi threaded. Ogni URL è manipolato da un thread chiamato ToeThread. Un ToeThread chiede al Frontier che gli sia assegnato un nuovo URL, lo invia attraverso i Processors, una volta terminate tutte le catene di processi richiede il successivo URL.

I Processors Sono raggruppati in catene di processori. Ogni catena esegue delle elaborazioni sull'URL assegnatoli da un ToeThread e, una volta completate, quest'ultimo passa l'URL alla catena successiva. Un Processor può far saltare una o più catene a un URL, per esempio in caso di errore.

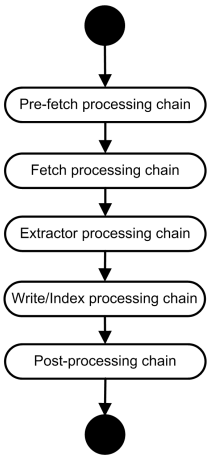

Figura 3.2.: Catene di processori di Heritrix

Le azioni eseguite dalle diverse processing chain sono le seguenti:

Pre-fetch processing chain La prima catena compie un controllo iniziale per verificare se l'URL può essere scansionato, controllando tutte le condizioni come per esempio DNS-lookup, standard robots.txt, autenticazione e la verifica di alcuni filtri configurabili. È possibile bloccare completamente il crawling degli URL che non hanno passato questo controllo iniziale.

Fetch processing chain In questa catena sono scaricati i dati dal server remoto, può essere presente un Processor per ogni protocollo supportato da Heritrix (per esempio http e dns).

Extractor processing chain A questo punto è disponibile il contenuto della pagina e i processori estraggono i link da essa.

Write/index processing chain Questa catena memorizza i dati nell'archivio, Heritrix di default utilizza ARCWriterProcessor, che li salva nel formato ARC. Possono essere scritti altri Processor che utilizzano formati e indici differenti.

Post-processing chain In quest'ultima catena sono aggiornate le informazioni sullo stato del crawling, vengono controllati i link estratti e memorizzati come CandidateURIs nel Frontier per essere successivamente scansionati.

Ogni URL è gestito con la classe *CrawURI*, nella quale sono memorizzati anche i link uscenti.

Heritrix è arrivato attualmente alla versione 1.14.4 (rilasciata il 10/05/2010), l'ultima versione può essere scaricata dal sito del progetto [2].

UbiCrawler

UbiCrawler è un web crawler scalabile, fault-tolerance, completamente distribuito, sviluppato presso l'università di Milano. Normalmente, anche in crawler ad architettura parallela, la gestione degli URL da visitare è gestita centralmente da un unico processo. In questo caso invece tutte le operazioni sono distribuite, anche quella di gestione-assegnazione degli URL da visitare, in questo modo si ottiene un'architettura molto robusta e fault-tolerance. Principali caratteristiche:

- Indipendente dalla piattaforma
- Ogni task è distribuito
- Assegnazione degli URL calcolata localmente basata su hashing consistente
- Tolleranza ai guasti
- Scalabilità
- Memorizzazione della struttura del grafo con WebGraph

Completamente distribuito Più agenti identici, identificati da un identificatore univoco, cooperano senza un coordinatore centrale. Ottenendo un sistema scalabile e facilmente configurabile. Non facendo alcuna assunzione sulla "localizzazione" degli agenti, la latenza può divenire un problema, quindi il sistema cerca di diminuire la comunicazione tra di essi.

Bilanciamento e assegnazione del lavoro calcolati localmente La distribuzione degli URL agli agenti che si occuperanno di visitarli è uno dei fattori critici: una buona gestione determina l'efficienza del processo di crawling. Sono stati rispettati tre requisiti:

- In ogni momento, ogni URL può essere assegnato a un agente, il quale diventa il solo responsabile per esso, in modo tale da diminuire la ridondanza dei dati.
- Ogni agente ha la capacità di identificare il responsabile di un URL senza comunicare con gli altri. In questo modo si riducono le comunicazioni, inoltre se un agente fallisce, l'assegnazione di un'URL a un altro agente ha

la capacità di trovare una nuova assegnazione senza nessuna comunicazione aggiuntiva.

- La distribuzione degli URL è bilanciata, in altre parole in caso di agenti omogenei, ogni agente avrà all'incirca lo stesso numero di URL, in caso di agenti eterogenei, gli URL saranno proporzionali alle risorse disponibili all'agente (memoria, dimensione disco, banda disponibile, ecc).

UbiCrawler è open source, ma non è distribuito online, per ottenerlo è necessario richiederlo ai responsabili del progetto presso l'università di Milano. Sono stati così contattati i responsabili (grazie al professor Luca Pretto), i quali ci hanno informati che il software non più mantenuto dal 2005, la relativa documentazione è scarsa e la sua configurazione è piuttosto difficoltosa.

Un altro prodotto open source disponibile sulla rete, è WebGraph, il quale non è un crawler, bensì uno strumento per la memorizzazione e lo studio del grafo del web. Di seguito saranno illustrate le sue principali caratteristiche e funzionalità.

WebGraph

Permette di gestire in maniera semplice grafi molto estesi, utilizzando moderne tecniche di compressione, e permette l'accesso e l'analisi dei grafi anche con PC poco potenti. WebGraph è composto di:

- Un insieme di codici uniformi, chiamati codici *theta*, che sono particolarmente adatti per l'archiviazione di grafi del web. L'effettiva efficacia di questi codici può essere provata sia empiricamente, sia attraverso un'accurata analisi matematica.

- Algoritmi per la compressione dei grafi del web che sfruttano gap compression e referenziazione, intervallizazione e codici *theta* per fornire un elevato rapporto di compressione: per esempio, il grafo WebBase del 2001 è compresso a 3,08 bit per link, e uno snapshot di circa 18.500.000 pagine del dominio ".uk", raccolte da UbiCrawler, è compresso a 2,22 bit per link. Gli algoritmi sono controllati da diversi parametri, che forniscono diversi compromessi tra velocità di accesso e rapporto di compressione.

- Algoritmi per accedere al grafo compresso senza decomprimerlo interamente, usando tecniche che rimandano la decompressione a quando è effettivamente necessaria.

- Una completa implementazione degli algoritmi sopra esposti in Java, contenuti nel package it.unimi.dsi.webgraph. Oltre alle API, il package contiene diverse classi che permettono di modificare o re comprimere un grafo.

- Il pacchetto si basa su: fastutil 5 per un type-specific, high-performance framework per collezioni, su MG4J per un I/O bit-level, sulla distribuzione COLT ready-to-use, su algoritmi efficienti e su JSAP per un parsing da linea di comando.

WebGraph è distribuito separatamente a UbiCrawler, l'ultima versione può essere scaricata dal sito del progetto [4]. All'interno dello stesso sito è possibile trovare diversi snapshot del dominio inglese e italiano, liberamente scaricabili.

3.3.3. Vincoli

Tecnologici: per la realizzazione del crawling è necessario utilizzare la linea messa a disposizione dal laboratorio che ospiterà le macchine (40Mbit/s condivisa tra tutte le utenze della sede e limitata ad un valore imposto dagli amministratori). Un ulteriore vincolo è dato dal rispetto delle politiche di visita fissate dai server web (attraverso il Robots Exclusion Protocol), il quale applica restrizioni all'analisi dei contenuti ospitati da tali server, indicando per esempio le pagine che non devono essere scansionate e il numero di visite per unità di tempo; alcuni server bannano i client che non rispettano tali politiche, in particolare quelle riguardanti gli accessi per unità di tempo. Il crawler dovrà quindi rispettare tali restrizioni.

Temporali: il tempo necessario al crawling deve essere il più breve possibile, poiché è molto importante ottenere il prima possibile lo snapshot del web italiano, per poter poi proseguire con la successiva analisi di quest'ultimo.

3.4. Ipotesi di lavoro

In questa sezione sono effettuate alcune considerazioni sui prodotti e i vincoli analizzati nelle sezioni precedenti, sono esposte le ipotesi di soluzione realizzabili e infine sarà motivata la scelta della soluzione da implementare.

3.4.1. Considerazioni sul mercato

UbiCrawler È l'unico tra quelli analizzati che integra al suo interno moduli per la gestione e memorizzazione del grafo ed è completamente distribuito. UbiCrawler è però un prodotto non più mantenuto, di difficile configurazione e con scarsa documentazione; lo studio necessario per prendere confidenza col prodotto e la successiva implementazione degli adattamenti necessari richiederanno quindi, molto più tempo (in quanto la documentazione è limitata), rispetto allo sviluppo dei soli moduli per la memorizzazione del grafo, necessari per le soluzioni implementate con Methanol e Heritrix, prodotti aggiornati e ben documentati.

Methanol È un prodotto più adatto ad essere utilizzato come motore di ricerca personale o di supporto ad un sito web. Permette però di distribuire le sue componenti su macchine differenti, aumentando così la sua scalabilità, è corredato da una dettagliata documentazione e supportato da una buona comunità di sviluppo. Manca però dei moduli per la memorizzazione del grafo.

Heritrix Heritrix è crawler creato per ottenere snapshot periodici di domini anche piuttosto estesi e meno adatto per essere utilizzato a supporto di un motore di ricerca, quindi si presenta come il prodotto più specifico per le esigenze rilevate da soddisfare, rispetto gli altri prodotti analizzati in questo studio. Presenta un'interfaccia web per la sua configurazione molto semplice da utilizzare, è disponibile una dettagliata documentazione ed è supportato da una buona comunità di sviluppatori, inoltre è il prodotto più aggiornato. Manca però dei moduli per la memorizzazione del grafo.

3.4.2. Considerazioni sui vincoli

I vincoli temporali impongono di scartare a priori l'implementazione della soluzione che prevede lo sviluppo di un nuovo applicativo, poiché richiederebbe tempi di realizzazioni troppo lunghi, inoltre presenta un fattore di rischio molto elevato, in quanto le problematiche da affrontare sono molteplici. Anche la soluzione che prevede l'adozione di UbiCrawler è stata scartata poiché prevedeva tempo di sviluppo e rischi superiori, rispetto alle altre due soluzioni rimanenti, causati dalla necessità di adattare e aggiornare un prodotto non mantenuto e con scarsa documentazione. Inoltre, la sua adozione nel progetto è stata fortemente sconsigliata dai suoi sviluppatori presso l'università di Milano.

È importante ridurre sia il tempo necessario al crawling, utilizzando un crawler efficiente, sia quello di sviluppo, utilizzando soluzioni semplici ed efficaci, riducendo possibilmente anche il carico di lavoro del sistema, così da ridurre i requisiti hardware necessari. Per quanto riguarda i vincoli sul Robots Exclusion Protocol, tutti i crawler presi in analisi implementano tecniche per rispettare tale standard. Esiste poi un vincolo relativo al software d'ambiente riguardante Heritrix e Methanol, il cui corretto funzionamento è garantito solo sotto Linux.

3.4.3. Ipotesi di soluzione

Sono ora esposti obiettivi e fasi di realizzazione delle ipotesi di soluzione valutate.

3.4.3.1. Obiettivi

L'obiettivo principale da perseguire è ottenere il grafo nel minor tempo possibile.

La scelta del software applicativo, tra Methanol e Heritrix, non differenzia molto le due ipotesi di soluzione. Esporremo quindi un'unica ipotesi, analizzeremo poi le differenze tra i due crawler per scegliere quello più adatto a questo progetto.

La soluzione prevede l'adattamento del software applicativo per permettere la creazione del grafo sviluppando, uno o più moduli, che si andranno a integrare al crawler, sfruttando la modularità offerta da entrambi i prodotti.

3.4.3.2. Fasi di realizzazione

a) Reperimento e studio di Heritrix/Methanol

b) Sviluppo del modulo necessario alla creazione grafo e interfacciamento con l'applicativo

c) Installazione

d) Test delle prestazioni nelle differenti configurazioni hw/sw ed eventuali adattamenti della configurazione hardware

e) Configurazione definitiva del sistema

f) Esecuzione del crawling

3.4.4. Aspetti particolari o critici

Dovendo utilizzare le risorse offerte dal laboratorio, si dovrà fare particolare attenzione alle prestazioni, poiché la velocità con cui un crawler scansiona le pagine, dipende in primo luogo dalla banda (la quale come esposto in precedenza sarà limitata), in secondo luogo dalle prestazioni dei dischi e se il crawler è distribuito dal numero di PC.

3.4.5. Analisi del rischio

Implementando una soluzione utilizzando Heritrix o Methanol il rischio è minore, perché sono prodotti già collaudati e specifici per questo scopo. Infine UbiCrawler non essendo più mantenuto, con scarsa documentazione e di difficile configurazione, comporterebbe un rischio troppo alto per il progetto, motivo per il quale la soluzione che lo utilizzava è stata scartata.

3.4.6. Soluzioni da valutare

Le soluzioni da valutare, come già detto in precedenza, si differenziano dal software per il crawling adottato. La soluzione che sarà implementata utilizzerà Heritrix come crawler, poiché è il software più indicato per il progetto: il suo campo di applicazione è più specifico, la documentazione è dettagliata e completa, l'interfaccia web è semplice ma permette di configurare la quasi totalità delle impostazioni ed è possibile sfruttare il supporto della comunità di sviluppatori.

3.5. Progetto di massima

È ora esposto il progetto di massima per la realizzazione del sistema di crawling.

3.5.1. Obiettivi

Adattare la struttura di Heritrix alle esigenze specifiche del progetto sopra esposte.

3.5.2. Architettura software

Secondo quanto detto in precedenza l'architettura software sarà composta dal crawler Heritrix, integrato con il modulo che permetterà la memorizzazione su disco del grafo. Verrà utilizzata una distribuzione di linux come software di ambiente.

3.5.3. Base di dati

Tutti i dati saranno memorizzati su di una struttura, che permetta di associare a ogni URL i link uscenti, ottenuti analizzando il documento scaricato da tale URL.

3.5.4. Componenti tecnologiche

Le componenti tecnologiche per la realizzazione del progetto sono: il server su cui far eseguire il sistema e una connessione a banda larga il più possibile performante.

Il dimensionamento della macchina sarà definito in seguito alla fase di test, i quali saranno eseguiti sul server kuma, che dispone di due processori dual core AMD e 4GB di ram.

3.5.5. Piano di realizzazione

La realizzazione del progetto inizierà con un'analisi più approfondita di Heritrix, per ottenere la giusta configurazione e poter progettare i moduli aggiuntivi per la scrittura del grafo. Una volta sviluppati e integrati i moduli aggiuntivi e scelta la configurazione ottimale si procederà con una serie di test per ottenere le informazioni necessarie al dimensionamento della macchina che sarà dedicata al crawling. Infine sarà lanciato il crawling e ne sarà monitorato il funzionamento.

3.5.6. Valutazione tempi di esecuzione

Con i dati reperibili in rete è stato possibile stilare alcune stime sui tempi di esecuzione. Non essendo disponibili informazioni aggiornate sulle dimensioni del dominio ".it", è stato necessario sviluppare le stime utilizzando i dati del dominio .it del 2004 e 2006, del dominio inglese (.uk), e relativi al numero dei domini .it registrati dal 2004 al 2009. In tutte le stime è stata utilizzata una dimensione media a nodo/documento di 50KB. Ora saranno esposte le tre stime che utilizzano le tre diverse fonti di dati.

Stima 1

Per il dominio .it sono disponibili solamente le seguenti informazioni [5]:

- .it 2004 41,3M pagine
- .it 2006 80M pagine, 193,7096% rispetto al 2004

con lo stesso tasso di crescita si ottiene nel 2010 un dominio di 288,8 milioni di pagine (13,45 TB).

Stima 2

Analizzando i dati relativi alla crescita del dominio .uk [5], riportati qui sotto, si può vedere come la crescita del web non sia lineare.

- .uk 2002 18,5M
- .uk 2005 50,6M, 273,5% rispetto 2002 (con 23,73 archi/nodo)
- .uk 2006 77,74M, 153,6% rispetto 2005 (con 31,5 archi/nodo)
- .uk 2007 105,9M, 136,2% rispetto 2006 (con 34,5 archi/nodo)

Stimando un andamento simile per il dominio .it, partendo dai dati relativi a tale dominio nel 2006, si otterrebbe oggi un dominio di 370 milioni di pagine (17,23 TB). Si può inoltre presumere che la media attuale di archi per nodo si attesti sui 43 archi/nodo.

Stima 3

Se si utilizza, come tasso di crescita delle pagine, il tasso di crescita dei domini .it registrati all'anno, fornito da Registro.it [3] (network information center italiano), si ottiene una dimensione dell'attuale dominio di 154,49 milioni di nodi (7,19TB), sempre partendo dai dati relativi al dominio ".it" del 2006.

anno	domini registrati	tasso di crescita %
2004	997261	
2005	1148273	115,14
2006	1296276	112,88
2007	1474194	113,72
2008	1623705	110,14
2009	1783178	109,82

Tabella 3.1.: Andamento annuale domini registrati (.it)

La media delle tre stime si attesta attorno ai 271,1 milioni di pagine, per un totale di 12,62 TB di documenti da scaricare.

Durante i test preliminari con Heritrix, la velocità media di download è stata di 3,4 MB/s (27,85 Mbit/s), con punte di 4,4 MB/s (37,4 Mbit/s). Con questa velocità sono necessari circa 83 giorni per scaricare tutti i dati, escludendo il tempo necessario per i checkpoint periodici, eseguiti giornalmente, i quali possono durare anche più di un'ora (durante il quale il crawling è fermo), quando si supera la decina di milioni di pagine scansionate. Con una stima di un'ora al giorno per i checkpoint otteniamo un tempo di crawling di 86,5 giorni.

Se si stima il tempo utilizzando la media degli URI scansionati al secondo, con una media di 90 URI/s (raggiunta con una configurazione di prova) si ottiene una stima di 34,86 giorni, i quali diventano 36,37 considerando i checkpoint.

Per concludere la stima a mio parere più probabile è quella ottenuta utilizzando la media di URI scansionati al secondo, poichè il numero di URL scansionati al secondo non è strettamete legato alla velocità di download (quindi alla banda disponibile), ma al numero di ToeThread e alle prestazioni della macchina in cui è eseguito il crawling. Bisogna tenere però conto del fatto che le medie di velocità sono state ottenute utilizzando tutta la banda disponibile nelle ore notturne; quando sarà lanciato il crawling, la banda a disposizione dovrà essere limitata (a un valore che sarà comunicato dai sistemisti), quindi le medie potrebbero diminuire.

3.6. Conclusioni

In conclusione si è deciso di realizzare il crawling utilizzando Heritrix, prodotto open source, integrandolo con un modulo per la memorizzazione del grafo, funzione non implementata nativamente nell'applicativo scelto. La scelta di questa soluzione è stata determinata dal fatto che richiedeva il minor tempo per l'implementazione e presentava il profilo di rischio minore per il successo del progetto.

4. Progettazione dell'applicativo

In questo capitolo saranno esposte diverse soluzioni analizzate per la realizzazione del modulo di gestione del grafo. Analizzeremo con maggior dettaglio il funzionamento di Heritrix, per poter meglio capire come sono state sviluppate e integrate le soluzioni all'interno del crawler, per garantire il corretto funzionamento del sistema nel suo complesso.

4.1. Panoramica Heritrix

Daremo ora una breve panoramica sul funzionamento di Heritrix.

Heritrix gestisce i crawling attraverso i *job*, ai quali sono associate tutte le impostazioni, i moduli e i sottomoduli da utilizzare. Può essere eseguito un solo job per volta.

Heritrix utilizza oggetti di tipo *CrawlURI* per memorizzare le informazioni relative ai documenti scansionati; quelle d'interesse per quest'analisi sono l'URL del documento e la lista dei link, estratti da esso. Heritrix è multithread, ogni thread elabora un CrawlURI, che richiede di volta in volta al *Frontier*. Il Frontier è il processo che tiene conto degli URI scoperti, quelli che si stanno processando, quelli processati e l'ordine con cui devono essere processati quelli in coda. Ogni thread è gestito con la classe *ToeThread*, la quale passa il CrawlURI assegnatoli, alle catene di processori chiamati anche moduli. Ogni processore implementa l'interfaccia Processor la quale presenta tre metodi rilevanti per l' analisi: *initialTasks(), innerProcess(CrawlURI)* e *finalTasks()*.

Per memorizzare il grafo sarà sviluppato un modulo. Analizzeremo ora le diverse catene di processori, per capire all'interno della quale, dovrà inserirsi tale modulo. Le catene nell'ordine:

Pre-fetch La prima catena compie un controllo iniziale di verifica sugli URL da scansionare, controllando tutte le condizioni impostate, come per esempio DNS-lookup, standard robots.txt, autenticazione e verifica alcuni filtri configurabili. È possibile scartare gli URL che non soddisfano tali condizioni.

Fetch In questa catena sono scaricati i dati dal server remoto. Può essere presente un modulo per ogni protocollo supportato da Heritrix (per esempio http, dns).

Extractor I processori di questa catena eseguono il parsing del contenuto del documento per estrarre i link. È presente un processore per ogni tipo di documento scansionato: html, pdf, doc, swf.

Write/Index In questa catena sono presenti i moduli che si occupano di memorizzare i dati relativi ai documenti scansionati. Heritrix, di default, utilizza il formato ARC per memorizzare informazioni riguardanti il contenuto dei documenti.

Post-processing In quest'ultima catena sono aggiornate le informazioni sullo stato del crawling, controllati i link estratti e memorizzati come CandidateURI nel Frontier per essere successivamente scansionati.

Nella catena di pre-selector, pre-fetcher e fetcher, saranno aggiunti dei filtri per limitare il crawling ai soli domini .it. Nello *Scope* sarà invece inserito, un filtro per escludere dalla scansione i file multimediali, quali immagini, video e file audio, così da ridurre la quantità di dati da scaricare. Questi file infatti, non contengono link al loro interno. Lo Scope si comporta come un filtro, il quale utilizzando le informazioni disponibili sui CandidateURI, decide l'ordine con cui devono essere scansionati.

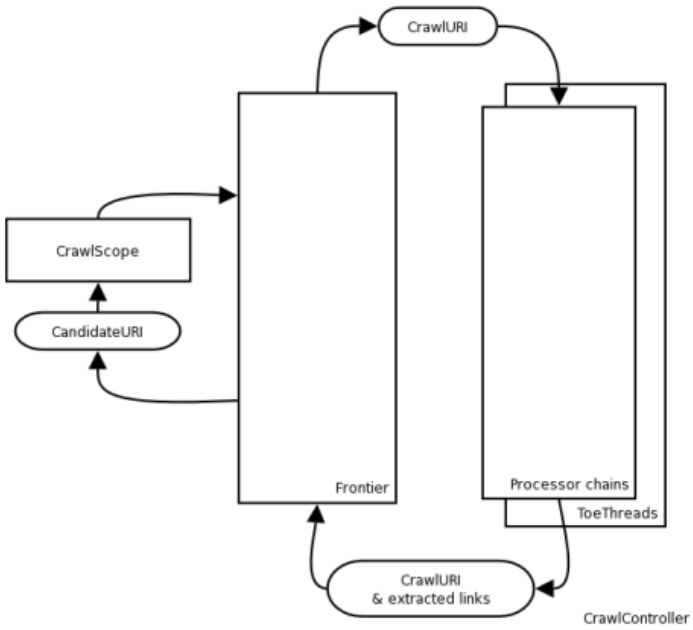

Figura 4.1.: Panoramica Heritrix

In un primo momento si può pensare di utilizzare la catena Write. Il processore da sviluppare utilizza l'oggetto di tipo CrawlURI per ottenere l'URL del

documento e la lista dei link uscenti. In questo modo però, si ottengono anche i link di domini non .it, poiché non è eseguito alcun controllo sui link estratti. Nella catena di post-processing i link uscenti sono controllati e inseriti in una lista chiamata *CandidateURI*, la quale contiene solo link validi che verificano i filtri impostati (nello Scope), nel nostro caso quelli relativi al dominio .it. Il nostro modulo si inserirà all'interno della catena di post-processin dopo i moduli di default, in modo di poter usufruire della lista dei CandidateURI, per memorizzare così le informazioni relative al nodo padre e agli archi uscenti da esso, cioè del URL visitato e dei link estratti dal documento.

Heritrix implementa una procedura per il backup utilizzando checkpoint periodici, dai quali in caso di crash è possibile riprendere il crawling con la minor perdita di dati possibile. La frequenza dei checkpoint è configurabile attraverso il file di configurazione delle proprietà di Heritrix. In caso di crash del sistema, se presente un checkpoint, è possibile riprendere il crawling creando un nuovo job dal recovery di quello interrotto. Nelle operazioni volte alla creazione del checkpoint sono messi in pausa tutti i thread prima di procedere con le operazioni di backup e una volta terminate, il crawling riprende la sua normale esecuzione.

4.2. Grafo

Analizzeremo ora le varie soluzioni d'implementazione del modulo per la memorizzazione del grafo e le relative strutture dati utilizzabili.

4.2.1. Soluzione 1: File ARC

Heritrix memorizza il contenuto dei documenti scansionati, all'interno di file ARC, uno per URL scansionato. I file ARC sono archivi compressi di soli file, non possono infatti contenere cartelle. Una volta terminato il crawling, possono essere utilizzati per ricostruire il grafo della porzione del web scansionato, analizzando il loro contenuto ed estraendo i link presenti nel documento. È quindi possibile creare il grafo in una fase post-crawling, ma questo richiederebbe tempo aggiuntivo e un notevole quantitativo di spazio per memorizzare tutti i documenti scaricati (qualche decina di terabyte), motivo per cui tale soluzione è stata scartata.

4.2.2. Soluzione 2: WebGraph

La seconda soluzione valutata prevede l'utilizzo di WebGraph per la memorizzazione del grafo. Per giungere alle motivazioni che ci hanno indotti a scartare anche questa ipotesi, analizzeremo in breve il funzionamento pratico di WebGraph.

Questo software implementa tecniche avanzate di compressione, per ridurre lo spazio necessario alla memorizzare su disco di un grafo del web. I nodi sono identificati da un interi a 32 bit. È possibile creare due tipologie di grafi: ImmutableGraph e MutableGraph. La prima permette di salvare su disco il grafo, ma non consente di aggiungere nodi e archi successivamente alla sua creazione, la quale avviene passando come parametro una matrice d'incidenza che rappresenta il grafo completo. La seconda invece, permette di inserire dinamicamente nodi e archi, successivamente alla creazione iniziale, ma non presenta metodi per il salvataggio su disco, per far ciò è necessario creare uno snapshot del grafo e convertirlo in ImmutableGraph. Nello snapshot creato non è però possibile inserire nodi e archi. È possibile fare il merge di due ImmutableGraph. Nel nostro caso quindi, è necessario utilizzare entrambe le tipologie: la prima per la memorizzare su disco, la seconda per inserire i nuovi nodi e archi scansionati. Inizialmente, il grafo sarà creato come MutableGraph. Periodicamente sarà creato lo snapshot per ottenere l'ImmutableGraph, il quale permetterà di effettuare il merge con l'eventuale grafo già presente su disco, in modo tale da ottenere alla conclusione del crawling il grafo completo. In seguito ad ogni snapshot del grafo si creerà un nuovo MutableGraph. Queste operazioni si rendono necessarie per due motivi: l'intero grafo non potrebbe risiedere interamente su ram e per motivi di sicurezza, se avvenisse un crash del sistema andrebbe perso l'intero grafo creato fino a quel momento.

WebGraph, come detto in precedenza, memorizza i nodi attraverso id numerici, ma non presenta metodi per associare un URL a un id. Per far ciò si utilizza un database o un file ordinato, in cui memorizzare la tupla URL-id. Per ottenere l'id associato ad ogni URL da inserire nel grafo, è necessario ricercare su disco la tupla corrispondente e se non presente crearne una nuova utilizzando id progressivi. Avendo stimato una media 53 archi per nodo (valore ottenuto analizzando un campione di 50 pagine e contando le occorrenze di "href" nel loro sorgente), si rendono necessari in media 54 accessi in lettura su disco per ogni URI processato (un accesso è necessario per ottenre l'id dell'URL processato). Con una media prevista di circa 60 URI/s, si effettuano 3240 accessi in lettura sul disco al secondo. Con un hard disk da 7.200 rpm, si ha un tempo di accesso medio di 8.9 ms e una latenza media di 4.17 ms, per un totale di 13.07 ms. Questi tempi permettono di effettuare in media 76.5 accessi in lettura al secondo, i quali non possono certo soddisfare le richieste necessarie ad implementare tale soluzione. È stata valutata quindi la possibilità di utilizzare SSD, i cui i tempi di accesso in lettura sono dell'ordine dei 0.2 ms e non avendo tempi di latenza permettono di effettuare 10000 accessi in lettura, riuscendo così a soddisfare la richieste stimate. Il problema in questo caso diventa il prezzo dei dischi che devono essere acquistati, non essendo disponibili in laboratorio, di circa 100 euro per 40 GB. Utilizzando

interi a 32 bit per gli id e stringhe di char per gli URL (1B per carattere e 64 caratteri in media per URL), otteniamo 4B+(64·1B) = 68B per tupla, con 271.1 milioni di documenti sono necessari 18 GB (20.22 con 300 milioni). Un singolo SSD da 40 GB sarebbe quindi sufficiente.

4.2.3. Soluzione 3: Funzione di hash e file su disco

L'ultima soluzione prevede l'utilizzo di una funzione di hash, per mappare gli URL negli id, e di due file per memorizzare il grafo. Questi ultimi hanno funzionalità differenti: il primo memorizza il grafo come sequenza di id del nodo padre e dei rispettivi figli, il secondo memorizza la lista degli URL scansionati.

La funzione di hash dovrà mappare in una sequenza di bit abbastanza lunga, per ottenere una probabilità di collisione estremamente bassa. Utilizzando il paradosso del compleanno, si può calcolare il numero minimo di bit necessari a ottenere una probabilità dello 0.5 che due codici diversi collidano.

$n = 2\log_2 271100000 = 56,029$

approssimando all'intero successivo si ottiene un numero minimo di bit pari a 29. Sempre per il paradosso del compleanno mappando a 128 bit, meno di 2^{29} URL, si ottiene una probabilità di collisione pari a 2^{-58}, circa 10^{-18}. Una probabilità estremamente bassa, adatta quindi al nostro caso, con grande probabilità infatti, non si verificheranno collisioni. Le funzioni di hash più diffuse sono MD5 (128bit) e SHA1 (160bit), riportiamo di seguito le loro prestazioni:

- MD5 1716Mbps ($13,4 \cdot 10^6$ digest al secondo)
- SHA1 609Mbps ($3,81 \cdot 10^6$ digest al secondo)

ottenute con un processore P4 a 2.1GHz. Entrambe soddisfano pienamente le richieste del software e non aggiungono un elevato carico di lavoro al processore. La funzione più adatta è MD5, poiché presenta un probabilità di collisione estremamente bassa (anche se inferiore a SHA1), un digest più corto e prestazioni in throughput superiori a SHA1.

Durante il crawling la funzione di hash restituisce gli id a 128bit degli URL, i quali sono poi inseriti nella sequenza binaria che compone il file del grafo. La distinzione tra il nodo padre e i nodi figli, avviene ponendo a uno il primo bit della sequenza del nodo padre, e a zero quello del nodo figlio. In questo modo viene eliminato un bit dal digest che diventa così a 127 bit, ma la probabilità di collisione resta pressoché invariata, sempre dell'ordine di 10^{-18}. All'interno della lista di URL saranno presenti ridondanze, poiché il controllo che evita di scansionare due volte lo stesso URL è effettuato prima di scansionare i documenti e non può essere eseguito sui link estratti dai questi (il controllo è eseguito confrontando il digest del CrawlURI attuale e quello dei CrawlURI precedentemente scansionati,

i link estratti non sono CrawlURI, ma sono *Link* in essi contenuti), si rende quindi necessario eliminare le ridondanze. Per far ciò, si suddivide la lista in due file: uno ordinato e privo di ridondanze, l'altro dove sono inseriti gli URL estratti durante il crawling, quindi non ordinato e con possibili ridondanze. Una volta al giorno è ordinato il secondo file e successivamente è eseguito un merge con il primo. Si riprenderà poi a scrivere la lista su di un nuovo file. Questa soluzione limita gli accessi in lettura al disco, a quelli necessari a ordinare il file e fare il merge, aumentando così le prestazioni del crawler. È possibile inoltre effettuare scritture utilizzando buffer di capienza opportuna, in modo tale da diminuire la frequenza della scrittura su disco, aumentando ulteriormente le prestazioni del sistema. Una volta terminato il crawling sarà associato a ogni URL il suo codice hash e un numero progressivo a 32 bit, che chiameremo *miniID*. Sarà eseguita quindi un'operazione di conversione per sostituire codici hash a 128 bit presenti nel grafo, con i corrispondenti miniID, una volta terminata l'operazione, si elimineranno i codici hash a 128 bit dalla lista di URL. Si manterrà la stessa convenzione, utilizzata precedentemente, per distinguere i nodi padri dai figli, cioè si pone a uno il primo dei 32 bit dei padri e a zero quello dei figli. Questo può essere realizzato poiché per numerare 271.1 milioni di nodi, bastano 29 bit, con i quali è possibile rappresentare più di 586 milioni di nodi. Dei restanti tre bit uno è utilizzato come appena descritto, gli altri due sono disponibili per usi futuri. Il grafo così costruito, permette di risparmiare il 75% percento dello spazio occupato originalmente.

È poi possibile utilizzare i miniID per creare un grafo con WebGraph, così da diminuire ulteriormente lo spazio occupato su disco e agevolare le successive operazioni di calcolo dei valori di PageRank, HITS e altri algoritmi.

Quest'ultima soluzione sarà quella implementata all'interno del modulo, poiché rispetto alla seconda soluzione non prevede l'acquisto di nuovo hardware e implementa una più semplice procedura di memorizzazione dei dati.

4.2.4. Checkpoint

Analizzeremo ora la soluzione utilizzata, per integrare alla gestione dei backup di Heritrix, quella relativa al grafo.

Progettando il modulo per la memorizzazione del grafo è stata considerata anche la sicurezza contro la perdita dei dati in seguito a crash del sistema. Se durante il crawling si verificasse un crash e non venisse gestito adeguatamente il ripristino dello stato del grafo, si creerebbero dei duplicati all'interno del grafo, cosa che si vuole fortemente evitare. Infatti, in seguito a un crash, lo stato di Heritrix viene ripristinato all'ultimo checkpoint disponibile, mentre lo stato del grafo rimane relativo all'istante precedente al crash. Quindi tutta la parte di grafo creata dal checkpoint al momento del crash è duplicata . Per risolvere questo

problema si è deciso di cambiare file di memorizzazione dei dati del grafo a ogni checkpoint, in questo modo, cancellando l'ultimo file del grafo e della lista di URL prima di eseguire il recovery, lo stato di Heritrix e del grafo sarebbero nuovamente allineati all'instante precedente al checkpoint, una volta eseguito il recovery. Alla fine del crawling si effettuerà il merge dei file del grafo, mentre i file della lista di URL saranno periodicamente ordinati e uniti in un unico file. Quest'ultima operazione, descritta anche in precedenza, permette di eliminare le ridondanze presenti nella lista, diminuendo così lo spazio richiesto per la memorizzazione.

Entrambi i tipi di file saranno numerati progressivamente e salvati in due cartelle differenti.

4.2.5. Progettazione classi Java

Le classi Java che saranno implementate sono tre: una si occupa di memorizzare il grafo come sequenza di byte, una per la memorizzazione della lista di URL come file di testo e infine il modulo che si integrerà con Heritrix e utilizzerà le altre due classi per memorizzare i dati. La prima classe BinaryFileWriterBuffred avrà un costruttore che richiederà come parametro il path della cartella che conterrà i file del grafo e opzionalmente la dimensione del buffer. All'interno del costruttore saranno inizializzati i buffer con il valore passato per parametro o con una capacità di default di 1800 sequenze di byte (con 60 URI/s sono necessari 30 s circa per riempire il buffer, quindi si scrive su disco ogni 30 s), ed è ricavato il numero progressivo per nominare il file (operazione necessaria per il recovery). La classe presenterà i seguenti metodi pubblici:

- write(byte[] sequece): inserisce una sequenza di byte nel file binario, ovvero la sequenza di bit del nodo padre e relativi nodi figli.
- close(): svuota il buffer e chiude il file su cui si stava scrivendo.
- changeNameToNext(): cambia il nome del file utilizzato, seguendo una numerazione progressiva.
- getFileName() e getFilePath(): i quali ritornano rispettivamente il nome e il path del file.

La seconda, UrlsFileWriterBuffered, è molto simile alla prima, con la differenza che scrive su file di testo e il metodo write accetta una stringa anziché una sequenza di byte.

In entrambe le classi sono utilizzati due buffer, in modo tale da non dover attendere che il buffer sia svuotato e scritto su disco, prima di riprendere la scrittura sul buffer. Scrittura su disco e su buffer sono operazioni svolte parallelamente, così da aumentare le prestazioni della classe.

Sarà implementata anche una versione senza buffer per ciascuna classe.

Il modulo GraphWriterBuffered, permetterà la configurazione dei valori delle dimensioni dei buffer e delle cartelle in cui memorizzare i file attraverso l'interfaccia web di Heritrix. Il modulo inizializza all'interno del metodo initialTask, richiamato una sola volta all'inizio del crawling, un'istanza di UrlFileWriterBufferd e una di BinaryFileWriter, utilizzate per la memorizzazione dei dati. Nel metodo innerProcess è calcolato il digest md5 di ciascun URL, creata la sequenza di bit e la lista di URL, le quali sono poi passate ai relativi metodi per la memorizzazione su disco. Nel metodo finalTask, richiamato alla fine del crawling, sono eseguite le operazioni per la chiusura dei file utilizzati per la memorizzazione del grafo. È stato inoltre implementato il metodo checkpoint(), che esegue le operazioni necessarie al backup dei file utilizzati per il grafo, richiamato durante la creazione del checkpoint all'interno della classe CrawlController, appositamente modificata.

5. Realizzazione e gestione del crawling

In questo capitolo saranno confrontati i risultati dei test nelle diverse configurazioni hardware e software, sarà trattato il dimensionamento del server, la configurazione di Heritrix e la soluzione ai problemi di rete riscontrati.

5.1. Test

Sono stati eseguiti alcuni test variando il numero di thread, la quantità di ram assegnata alla JVM e il numero di cpu, per analizzare il comportamento di Heritrix al variare delle configurazioni e poter poi dimensionare la macchina che lo ospiterà durante il crawling. Sono stati inoltre eseguiti test comparativi sui moduli sviluppati con e senza buffer.

5.1.1. Test scrittura su disco bufferizzata

Le prestazioni di un crawler, come già esposto più volte, possono essere limitate da quelle del sistema di storage utilizzato. Sono effettuati infatti, molti accessi in lettura e scrittura, ed è per questo motivo che le classi dedicate alla memorizzazione dei dati relativi al grafo, sono state sviluppate con un occhio di riguardo alle prestazioni. Sono riportati in Figura 5.1 i risultati dei test con e senza buffer.

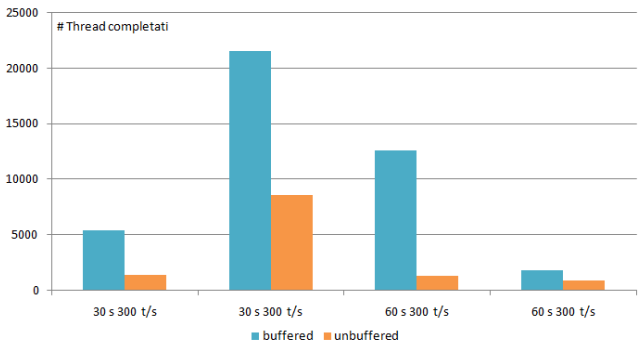

Figura 5.1.: Grafico comparativo test scrittura bufferizzata

Per questo test è stato realizzato un piccolo software che simula il comportamento di Heritrix, generando 300 thread al secondo che svolgono le stesse operazioni svolte dal modulo per la memorizzazione del grafo, compreso il calcolo del digest md5, così da emulare i ToeThread. Sono state effettuate quattro prove: due della durata di 30 secondi e due di 60 secondi.

Monitorando l'utilizzo dei dischi durante il crawling, è stato possibile confrontare la banda utilizzata da Heritrix con GraphWriter e GraphWriterBuffered. Si può osservare che l'utilizzo del primo modulo genera un flusso più consistente in scrittura (circa 1.5 MB/s in media, con picchi di 7.4 MB/s), mentre con il secondo modulo si ha un utilizzo continuo in scrittura molto più basso (circa 500 KB/s in media, con picchi di 11.6 MB/s). Questo significa che la versione bufferizzata sfrutta maggiormente le risorse di storage, ma per un minor lasso di tempo, lasciando così un maggior numero di operazioni su disco disponibili per altri moduli di Heritrix.

5.1.2. Test sul quantitativo di ram

Il primo test è stato effettuato modificando la quantità di ram assegnata alla JVM e mantenendo costante il numero di thread a 200, utilizzando il server svrhpcopt-1 con doppio processore AMD Opteron (single core), 8 GB di ram DDR, senza limitazioni di banda. La durata di ciascun test è di un'ora.

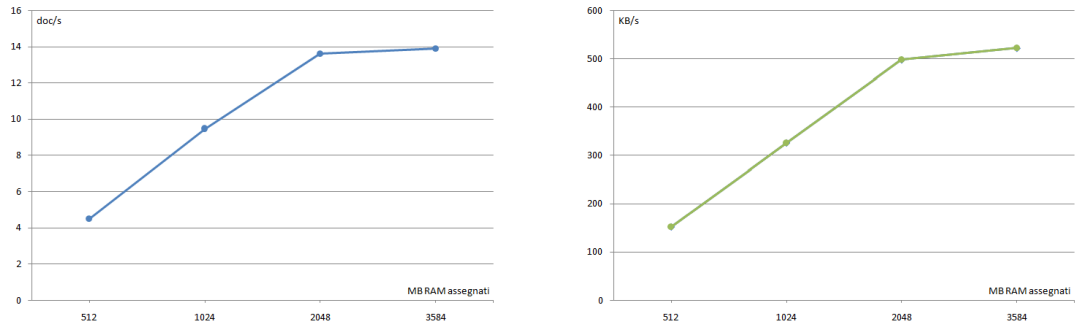

Figura 5.2.: Grafico della velocità di download (sinistra) e di elaborazione dei documenti (destra), test ram

Come si può notare l'aumento di ram comporta una diretta crescita delle prestazioni, l'ultimo test ha segnato un minor guadagno prestazionale, poiché

soli 200 thread non permettono di sfruttare completamente il quantitativo di ram a disposizione.

Il grafico seguente mette in relazione la ram utilizzata dal crawler con le sue prestazioni relativamente al numero medio di documenti scansionati al secondo e alla velocità media di download. Il test è stato effettuato su svrhpcopt-1, con una configurazione a 800 thread e 3328MB di ram assegnata alla JVM.

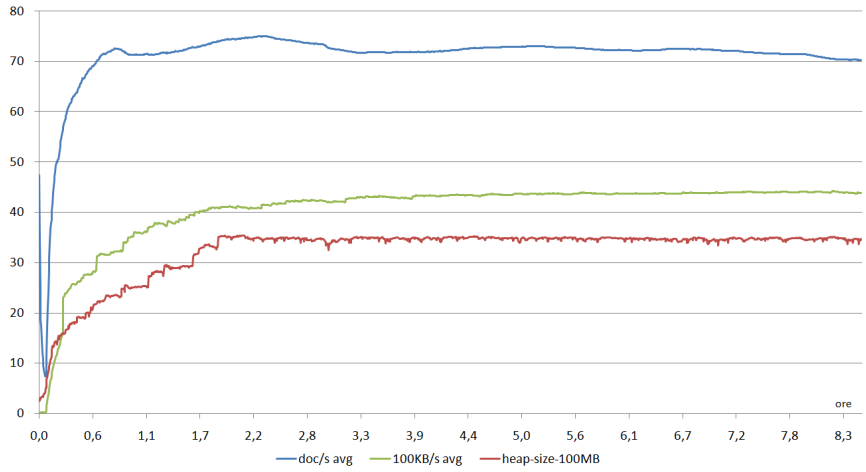

Figura 5.3.: Grafico velocità download e elaborazione documenti, test ram

Come si può notare dal grafico, l'andamento delle prestazioni è fortemente legato alla quantità di memoria disponibile alla JVM, infatti, la saturazione di questa interrompe l'aumento delle velocità di download e di elaborazione dei documenti, le quali diventano pressoché costanti.

È stato inoltre eseguito un test con 6 GB di ram e 1200 thread senza però ottenere significativi aumenti di prestazioni rispetto alla configurazione con 3,3 GB di ram e 800 thread.

5.1.3. Test sul numero di thread

La seconda serie di test sono stati condotti variando il numero di thread e lasciando invariata la quantità di ram assegnata alla JVM, a 3328MB. Il server utilizzato è svrhpcopt-1 e la durata dei test è di 30 minuti.

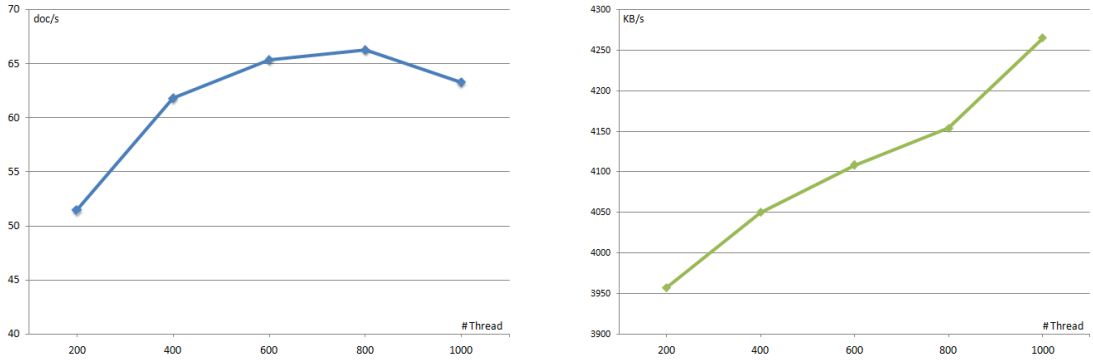

Figura 5.4.: Grafico della velocità di download (sinistra) e di elaborazione dei documenti (destra), test thread

Dai grafici è possibile osservare che l'aumento di thread comporta una proporzionale crescita della velocità di download. La velocità di elaborazione dei documenti invece presenta un diverso andamento: si ha un notevole beneficio con il primo incremento di thread, un beneficio minore con i due successivi, per ottenere poi un degradamento prestazionale nell'ultimo test, dovuto alla insufficiente disponibilità di memoria richiesta da un numero così elevato di thread.

5.1.4. Test cpu

I test per confrontare le prestazioni con una e due cpu, non sono stati portati a termine, poiché durante il test a due cpu su kuma (dual core e dual cpu), è stato saturato il firewall della struttura universitaria che ospitava il server. Si è reso quindi necessario interrompere il crawling per non creare disagi alle altre utenze della rete. Per risolvere questo problema e quello relativo alla limitazione della banda utilizzata (anch'essa spesso saturata in buona parte dei test effettuati), sono state implementate soluzioni esposte in 5.3.3.

Sono comunque disponibili alcuni dati approssimativi dopo alcune ore di crawling, con una configurazione a 800 thread e 3328 MB di memoria assegnata alla JVM.

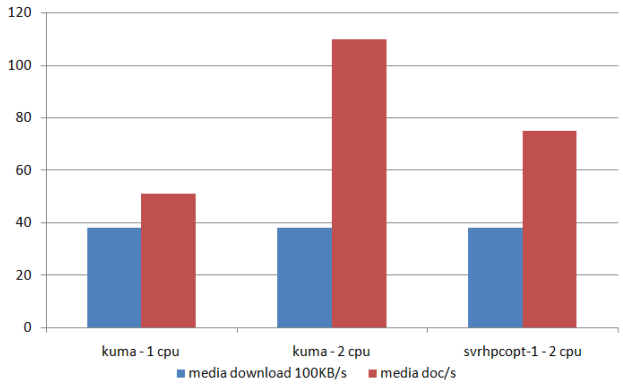

Figura 5.5.: Grafico confronto prestazioni a singolo o doppio processore

Dal grafico si evince che al variare della potenza di calcolo disponibile, pur rimanendo pressoché costante la velocità di download, il numero di documenti scansionati al secondo raddoppia al raddoppiare della potenza di calcolo.

5.2. Dimensionamento server

Sarà ora esposto il dimensionamento del server che ospiterà il crawling, per far ciò sarà stimato prima il numero medio di archi uscenti per nodo, per stimare la dimensione su disco del grafo, sarà poi stimato lo spazio su disco necessario a memorizzare le informazioni che Heritrix utilizza per portare a termine il crawling. Infine saranno definite le specifiche hardware complessive.

5.2.1. Stima numero di archi del grafo

Per stimare lo spazio richiesto per la memorizzazione, si è reso necessario avere una stima del numero medio di archi uscenti per nodo. La stima è stata effettuata utilizzando i dati reperibili in rete, in particolare quelli relativi ai dataset di WebGraph [5], riferiti al dominio inglese (.uk):

Crawl date	arc/node
2002	16,10
2005	23,73
2006	33,96
2007	35,33

Tabella 5.1.: Andamento numero di archi uscenti per nodo

Mantenendo lo stesso trend di crescita si ottiene la seguente previsione:

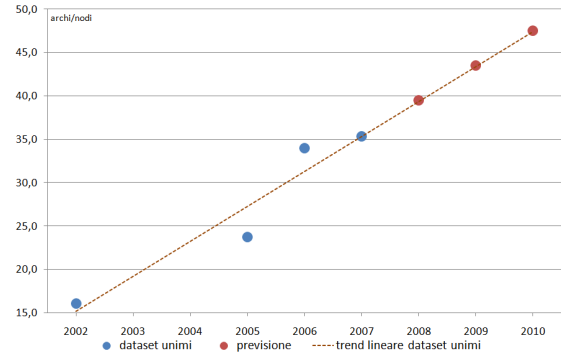

Figura 5.6.: Garfico andamento e previsione numero di archi uscenti per nodo

Ottenendo così nel 2010 un numero di archi per nodo pari a 48. Nel 2004 lo snapshot del web italiano presentava 27.87 archi/nodi (contro i 23.73 del dominio inglese nel 2005), facendo una proporzione con il dominio .uk si ottiene una media di 58 archi/nodi per il dominio .it nel 2010.

Sono stati poi condotti dei test utilizzando il modulo GraphWriterBuffered, il quale nel suo report mostra il numero medio di link per documento, media calcolata come rapporto tra il numero CrawlURI processati e link uscenti in essi riscontrati, il quale su test da 10 ore ha riportato una media di 51 link uscenti per CrawlURI, quindi 51 archi/nodo.

Per la stima dello spazio su disco sarà quindi utilizzato 54.5 archi/nodo, valor medio delle due stime.

5.2.2. Stima dimensione disco

Per stimare lo spazio su disco necessario a memorizzare i file utilizzati da Heritrix, sono stati utilizzati dati ottenuti con i precedenti test. La tabella seguente riporta i dati utilizzati:

	crawl 1	crawl 2	crawl 3
# URI scansionati	117686	893514	829631
log MB	56	419	386
state MB	0,004	2048	2600
scratch MB	768	882	56
checkpoint		2048	2600

Tabella 5.3.: Dati dimensione file su disco dei crawling

Da questi dati si può vedere che la dimensione del checkpoint è uguale a quella necessaria ai file di state, poiché ne è una copia. I file di scratch sono invece temporanei.

Per calcolare lo spazio su disco richiesto da Heritrix, è stata eseguita una proporzione tra spazio occupato e numero di URI scansionati, ottendo così i seguenti dati:

- 600 GB per i file di log e state
- 500 GB per 2 checkpoint
- 25 GB per i file di scratch

Per i file di scratch non è stata eseguita la stessa proporzione, poiché sono dati temporanei, i quali in genere diminuiscono molto in seguito ad un checkpoint ma la loro dimensione è molto variabile e di difficile stima. È stata fatta una proporzione con i dati presentati sopra e altri qui non riportati, utilizzando un numero di URI, calcolato in base a quelli scansionati al giorno ($5,184 \cdot 10^6$ URI/g, periodo di un checkpoint), e assumendo un margine di errore del 30%, data l'elevata incertezza.

Per quanto riguarda il grafo, come esposto precedentemente, utilizziamo id a 128 bit per ogni nodo, quindi con 271.1 milioni di nodi e 54.5 archi/nodo, otteniamo una dimensione finale di 220 GB. La lista di URL invece, usa caratteri da 8 bit con una media di 61 caratteri per URL, ottenuta come il rapporto tra numero di URL e caratteri presenti nel file degli URL dopo un crawling di 12 ore, si ottiene così 16GB.

Il totale risulta essere 1361 GB per memorizzare il grafo e le informazioni necessarie ad Heritrix.

5.2.3. Configurazione hardware complessiva

Analizzando i risultati dei precedenti test, si è deciso di utilizzare il server kuma equipaggiato con due cpu Athlon X2 7550 a 2.5 GHz, 4 GB di ram DDR2. Per lo storage si è optato per un raid 0 composto di quattro dischi da 500 GB per un totale di 2 TB utilizzato per memorizzare i file di Heritrix, mentre sistema operativo e crawler risiederanno su di un disco da 160 GB.

5.3. Configurazione sistema

Sono ora esposte le configurazioni del sistema operativo, del software d'ambiente e di Heritrix. Sono inoltre riportate le soluzioni implementate per risolvere i problemi di rete riscontrati.

5.3.1. Sistema operativo e software d'ambiente

Il sistema operativo utilizzato è Debian 2.6.32-5-amd64, con Java SE versione 1.6.0. La ram assegnata a Heritrix è di 3328MB, un quantitativo superiore comporterebbe il crash della macchina in seguito al raggiungimento del limite di memoria a disposizione (per "out of memory process impossible to kill"). Inoltre si è reso necessario modificare il limite del numero di file contemporanei aperti (/etc/security/limits.conf), impostandolo a 32768, altrimenti Heritrix termina prematuramente il crawling, a causa dell'errore "too many open files".

5.3.2. Configurazione Heritrix

La configurazione definitiva prevede:

- 800 thread
- 3328 MB di ram assegnata alla JVM
- utilizzo del modulo GraphWriterBuffered (dimensione dei buffer 1800)
- seed list composta di circa mille URL
- filtro per non scaricare le immagini
- filtri per limitare il crawling al solo dominio .it

Per informazioni dettagliate sui parametri di configurazione vedere l'appendice A.

5.3.2.1. Seed list

La seed list utilizzata contiene più di mille URL, di cui una ventina sono siti directory non appartenenti al dominio .it, si è reso quindi necessario applicare dei filtri che limitassero la scansione ai soli siti .it, ma allo stesso tempo rendessero possibile scansionare i siti directory della seed list. Il corretto funzionamento di questi filtri è stato testato con un sito creato appositamente, su un dominio non .it. È stato inoltre implementato un filtro all'interno del modulo GraphWriter-Buffered, per non scrivere su disco i dati relativi ai siti directory non .it, i quali non vengono eliminati dai filtri.

5.3.3. Problemi e soluzioni adottate per banda e numero di connessioni TCP

Sono ora trattate le soluzioni ai problemi di rete riscontrati.

5.3.3.1. Banda

Heritrix implementa un limite, all'interno del Frontier, sulla banda totale utilizzata, il quale è stato inizialmente settato a 1500KB/s. Questa impostazione provocava però un forte degrado prestazionale, poiché agisce sulla velocità media di download misurata da Heritrix, sospendendo i download quando questa supera il valore limite, causando così la diminuzione della media. In questo modo però, i download sono sospesi molto spesso e si provocano comunque picchi molto alti sulla banda utilizzata, diminuendo notevolmente la velocità di elaborazione dei documenti. Si è quindi deciso di non utilizzare questa opzione per limitare la banda, bensì *wondershaper*, software open source, che permette di limitare la banda su linux, con il quale le medie sono più che raddoppiate, rispetto a quelle ottenute con il limite imposto da Heritrix.

Analizzando il grafico settimanale che riporta l'utilizzo della banda nella struttura in cui sarà ospitato il server per il crawling, sono state riscontrate tre diverse condizioni di carico: di giorno in cui c'è molto traffico, di notte in cui si ha un carico ridotto e il sabato e la domenica in cui si ha un carico leggermente superiore rispetto alle ore notturne.

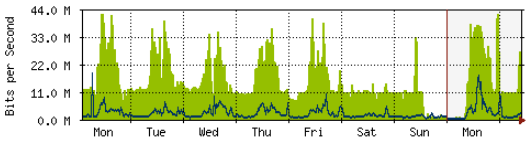

Figura 5.7.: Grafico settimanale utilizzo banda

Dal grafico è possibile osservare un utilizzo di fondo di circa 10 Mbit provocato dal crawler, costante per tutte le ore del giorno. Tra domenica e lunedì il crawler è stato fermato per manutenzione alla rete del laboratorio, potendo notare così il basso traffico presente la domenica e la notte. È stato quindi implementato uno script che modifica automaticamente il limite sulla banda (utilizzando wondershaper e *cron*), così da poter usufruire della maggior disponibilità di banda presente nelle diverse condizioni di carico.

5.3.3.2. Connessioni TCP contemporanee

Durante la saturazione del firewall erano aperte più di 12000 connessioni TCP. Si è reso necessario limitarle modificando il sorgente di Heritrix, poiché imponendo un limite a livello di sistema operativo, sarebbero state chiuse connessioni aperte dal crawler, il quale avrebbe potuto interpretare il fatto, come un errore di connessione e non scansionare gli URL le cui connessioni sono state forzatamente chiuse. Analizzando il codice è stata identificata la classe che gestisce le connessioni http, tramite la classe Socket della libreria standard di Java. Questa classe, httpConnection, quando chiude le connessioni non esegue alcun controllo sull'effettiva chiusura delle stesse, così facendo alcune connessioni rimangono aperte fino allo scadere del timeout, impostato a 20 s. Si crea così un gran numero di connessioni non utilizzate in attesa di essere effettivamente chiuse, perché per esempio non hanno ricevuto l'ultimo ack di chiusura. Con 800 thread attivi e una media di 110 URI/s, sono create più di 100 connessioni al secondo, in 20 secondi più 2220 connessioni le quali, se non sono chiuse in poco tempo, raggiungono un numero troppo elevato per essere gestite dal firewall della struttura universitaria. La classe è stata quindi modificata inserendo un controllo che impedisce l'apertura di nuove connessioni, se il numero di quelle ancora attive ha raggiunto il limite imposto. Il controllo del numero di connessioni attive è stato demandato a un thread, il quale per far ciò controlla l'effettiva chiusura delle connessioni, liberando così il metodo di chiusura delle connessioni da tale controllo. In questo modo si evita che un ToeThread che chiude la connessione attenda inutilmente il suo rilascio.

5.4. Conversione del grafo

Una volta terminato il crawling, è necessario trasformare il grafo in un formato il più possibile leggero e di facile elaborazione e manipolazione. Per far ciò è stato implementato un piccolo software che converte il grafo, sostitueno i nodi a 128

bit con nodi a 32 bit, e crea delle strutture dati che permettano di risalire il più velocemente possibile all'URL corrispondente ad ogni nodo e viceversa.

Il software è composto di due moduli. Il primo, parte dalla lista degli URL scansionati, ordinata alfabeticamente, e crea un file di testo temporaneo contenente la lista di tutti gli hash MD5 degli URL associandovi un id progressivo a 32 bit, chiamato *miniId*. Il file è poi ordinato in base all'hash, e ne viene creata una copia binaria, con campi a lunghezza fissa, la quale servirà poi per eseguirvi la ricerca dicotomica degli hash (nel file di testo le linee hanno dimensione variabile, non è quindi possibile effettuare tale ricerca, la quale è invece possibile in un file binario composto da campi di lunghezza prefissata). Infine, è letto ogni hash a 128 bit del grafo, cercando la corrispondenza nel file binario (con una ricerca dicotomica), e ottenendo così i miniId, utilizzati per creare la copia compressa del grafo. L'altro modulo crea un file per l'associazione *URL-miniId*, utilizzata per le successive fasi di analisi e elaborazione del grafo. I due file temporanei per l'associazione *hash-miniId* vengono eliminati alla fine delle elaborazioni.

Conclusioni

In questa relazione sono state vagliate le possibili soluzioni per la realizzazione e gestione di un sistema di crawling, con l'obiettivo di ottenere il grafo del web italiano, in particolare del dominio .it, nel minor tempo possibile e con il minimo dispendio di risorse. La soluzione implementata prevede l'utilizzo del crawler open source Heritrix, integrato con un modulo per l'ottenimento del grafo. La struttura dati per la memorizzazione del grafo è stata progettata al fine di ottenere un grafo il più possibile semplice da utilizzare nelle successive fasi del progetto, di cui il lavoro qui descritto è solo un modulo. Nella realizzazione del sistema sono state affrontate differenti problematiche per superare i diversi colli di bottiglia e le restrizioni imposte dalle risorse disponibili, in particolare quelle riguardanti la rete (banda e numero di connessioni TCP contemporanee). Un forte limite al raggiungimento di quest'obiettivo è stato imposto dalla limitata banda a disposizione e dal numero massimo di connessioni TCP aperte. La soluzione implementata per limitare la banda prevede l'utilizzo di uno script che varia dinamicamente il limite a seconda delle condizioni di carico, con valori differenti nelle ore diurne, notturne e nel fine settimana. Per limitare il numero di connessioni TCP contemporanee è stato implementato un controllore delle connessioni attive che impedisce l'apertura di nuove connessioni se il numero di quelle aperte ha raggiunto il limite imposto. Nonostante tutte le limitazioni, è stato realizzato un sistema di crawling che permette di ottenere il grafo della porzione del web scansionato, offrendo un ottimo strumento per reperire dati su cui operare studi e analisi nel campo dell'Information Retrieval.

Elenco delle figure

Elenco delle tabelle

A. Configurazione Heritrix

La versione di Heritrix utilizzata è la 1.14.4 modificata con la gestione delle connessioni e con l'aggiunta del modulo GraphWriterBuffered.

Sono riportate le impostazioni utilizzate per i settaggi di Heritrix:

Moduli:

Crawl Scope:

- *org.archive.crawler.scope.SurtPrefixScope*

URI Frontier:

- *org.archive.crawler-BdbFrontier*

Pre Precessors:

- *org.archive.crawler.prefetch.Preselector*
- *org.archive.crawler.prefetch.PrecodintionEnforcer*

Fetcher:

- *org.archive.crawler.fetcher.FetchDNS*
- *org.archive.crawler.FetchHTTP*

Extractors:

- *org.archive.crawler.extractor.ChangeEvaluator*
- *org.archive.crawler.extractor.ExtractorHTTP*
- *org.archive.crawler.extractor.ExtractorHTML*
- *org.archive.crawler.extractor.ExtractorCSS*
- *org.archive.crawler.extractor.ExtractorJS*

- *org.archive.crawler.extractor.ExtractorSWF*
- *org.archive.crawler.extractor.ExtractorPDF*
- *org.archive.crawler.extractor.ExtractorXML*
- *org.archive.crawler.extractor.ExtractorDOC*
- *org.archive.crawler.extractor.ExtractorURI*

Writer:

- non sono stati utilizzati

Post Processor:

- org.archive.crawler.postprocessor.CrawlStateUpdater
- org.archive.crawler.postprocessor.LinksScoper
- org.archive.crawler.postprocessor.FrontierScheduler
- org.archive.crawler.postprocessor.LowDiskPauseProcessor
- org.archive.crawler.postprocessor.GraphWriterBuffered
- org.archive.crawler.postprocessor.SupplementaryLinks

Sottomoduli:

scope:exclude-filter:filter:

- *org.archive.crawler.filter.URIRegExpFilter*

uri-canonicalization-rules:

- Lowercase = *org.archive.crawler.url.canonicalize.LowercaseRule*
- Userinfo = *org.archive.crawler.url.canonicalize.StripUserinfoRule*
- WWW[0-9]* = *org.archive.crawler.url.canonicalize.StripWWWNRule*
- SessionIDs = *org.archive.crawler.url.canonicalize.StripSessionIDs*
- SessionCFIDs = *org.archive.crawler.url.canonicalize.StripSessionCFIDs*
- QueryStrPrefix = *org.archive.crawler.url.canonicalize.FixupQueryStr*

org.archive.crawler.deciderules.SurtPrefixedDecideRule apllicato a:

- Preselector#decide-rules
- Preprocessor#decide-rules
- DNS#decide-rules
- HTTP#decide-rules
- ExtractorHTTP#decide-rules
- ExtractorCSS
- ExtractorJS#decide-rules
- ExtractorSWF#decide-rules
- ExtractorPDF#decide-rules
- ExtractorXML#decide-rules
- ExtractorDOC#decide-rules
- ExtractorURI#decide-rules

Impostazioni: Sono riportate solo le impostazioni non di default.
max-toe-thread: 800
filters:escludiImm:

- enable:True
- if-match-return:True
- regexp: .*(?i).(bmp|cab|gif|jpe|jpg|jpeg|png|tiff|mid|mp2|mp3|mp4|wav|avi|mov|mpeg|ram|rm|smil|wmv|ppt)$

rules: SurtPrefixedDecideRule:

- decision:Accept
- seeds-as-surt-prefixes:True
- also-check-via: False
- rebuild-on-reconfig: True

midfetch-decide-rules:

- max-contemporary-tcp-connection: 5000

GraphWriterBuffered:

- graph-size-buffer: 1800
- urls-size-buffer: 1800

La dimensione dei buffer corrisponde al numero di CrawlURI processati tra una scrittura su disco e quella successiva. Utilizzando la velocità media di elaborazione dei documenti è possibile calcolare la dimensione del buffer definendo ogni quanti secondi eseguire la scrittura su disco (svuotando il buffer).

Dimensione del buffer = (intervallo tra due scritture $[s]$) $\cdot$ (media $[\text{doc}/s]$)

B. Guida all'utilizzo del software di conversione del grafo

I file per la conversione del grafo e per la creazione del file per l'associazione *URL-id* si trovano nella cartella principale del pacchetto makeGraph.

È necessario impostare alcuni parametri nel file makeGraphStructure.sh.

- URLS_FILE: indirizzo del file completo e ordinato di tutti gli URL
- GRAPH16B_FILE: indirizzo della cartella contenente i file del grafo prodotti da Heritrix
- GRAPH4B_FILE: nome del file del grafo che sarà creato
- OUT_DIR: indirizzo della cartella in cui saranno salvati il grafo e la lista URL-id

Infine è sufficiente eseguire lo script per ottenere la conversione del grafo e la lista URL-miniId, che saranno salvati nella cartella specificata nel parametro OUT_DIR.

Bibliografia

[1] Methanol Web Crawling System, http://metha-sys.org. Last visit September 2010.

[2] Progetto Heritrix, http://crawler.archive.org/. Last visit September 2010.

[3] Registro.it - Annuari, http://www.nic.it/tutto-sul.it/annuario-dati-del-registro.it. Last visit May 2010.

[4] WebGraph, http://webgraph.dsi.unimi.it/. Last visit September 2010.

[5] WebGraph Dataset, http://law.dsi.unimi.it/index.php?option=com_include&Itemid=65. Last visit May 2010.

[6] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.

[7] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, pages 711–726, 2004.

[8] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proc. of WWW*, pages 595–601, 2004.

[9] C. Castillo and R. Baeza-Yates. Wire: an open-source web information retrieval environment. In *Workshop on OSWIR*, pages 27–30, 2005.

[10] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, pages 1623–1640, 1999.

[11] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proc. of VLDB*, pages 200–209, 2000.

[12] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proc. of the ACM SIGMOD*, pages 117–128, May 2000.

[13] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through url ordering. In *Proc. of WWW*, pages 161–172, 1998.

[14] M. Diligenti, F. Coetzee, S. Lawrence, L. C. Giles, and M. Gori. Focused crawling using context graphs. In *Proc. of VLDB*, pages 527–534, September 2000.

[15] S. Dill, R. Kumar, K. S. Mccurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins. Self-similarity in the web. *ACM TOIT*, pages 205–223, 2002.

[16] J. E. H. e Gordon Mohr e Kristinn Sigurdsson e Michael Stack. *Heritrix developer documentation.* Internet Archive. http://crawler.archive.org/articles/developer_manual/index.html, 2008.

[17] K. S. e Michael Stack e Igor Ranitovic. *Heritrix User Manual.* Internet Archive. http://crawler.archive.org/articles/user_manual/index.html, 2008.

[18] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. Measuring index quality using random walks on the web. In *Proc. of WWW*, pages 1291–1303, 1999.

[19] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform url sampling. In *Proc. of WWW*, pages 295–308, 2000.

[20] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, pages 604–632, 1999.

[21] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2009.

[22] M. Najork. Breadth-first search crawling yields high-quality pages. In *Proc. of WWW*, pages 114–118, 2001.

[23] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proc. of VLDB*, pages 129–138, 2001.

[24] J. Rennie and A. K. Mccallum. Using reinforcement learning to spider the web efficiently. In *Proc. of ICML*, pages 335–343, 1999.

[25] B.-Y. Ricardo and C. Carlos. Link analysis in national web domains. In *Proc. of OSWIR*, pages 15–18, 2005.

[26] E. Romanus. *Methanol Web Crawling System 1.7.0.* http://metha-sys.org/manual/html_node/index.html, 2008.

[27] B. Sergey and P. Lawrence. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 2006.

[28] V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed web crawler. In *Proc. of IEEE ICDE*, 2002.

[29] C. Soumen, C. Soumen, and C. Soumen. Distributed hypertext resource discovery through examples. In *Proc. of VLDB*, pages pages 375–386, September 1999.

[30] Z. B. Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about web pages via random walks. In *Proc. of VLDB*, pages 535–544, 2000.