Search Engines and Web Dynamics

Knut Magne Risvik
Fast Search & Transfer ASA
Knut.Risvik@fast.no

Rolf Michelsen
Fast Search & Transfer ASA
Rolf.Michelsen@fast.no

Abstract

In this paper we study several dimensions of web dynamics in the context of large-scale Internet search engines. Both growth and update dynamics clearly represent big challenges for search engines. We show how the problems arise in all components of a reference search engine model.

*Furthermore, we use the FAST Search Engine architecture as a case study for showing some possible solutions for web dynamics and search engines. The focus is to demonstrate solutions that work in practice for real systems. The service is running live at **www.alltheweb.com** and major portals worldwide with more than 30 million queries a day, about 700 million full-text documents, a crawl base of 1.8 billion documents, updated every 11 days, at a rate of 400 documents/second.*

We discuss future evolution of the web, and some important issues for search engines will be scheduling and query execution as well as increasingly heterogeneous architectures to handle the dynamic web.

1 Introduction

Search Engines have grown into by far the most popular way for navigating the web. The evolution of search engines started with the static web and relatively simple tools such as WWWW [McB94]. In 1995 AltaVista launched and created a bigger focus on search engines [SRR97]. The marketplace for search engines is still dynamic, and actors like FAST (www.alltheweb.com), Google, Inktomi and AltaVista are still working on different technical solutions and business models in order to make a viable business, including paid inclusion, paid positioning, advertisements, OEM searching, etc.

A large number of analyses have been made on the structure and dynamics of the web itself. Conclusions are drawn that the web is still growing at a high pace, and the dynamics of the web is shifting. More and more dynamic and real-time information is made available on the web. The dynamics of the web creates a set of tough challenges for all search engines.

In Section 2 we define a reference model for Internet search engines. In Section 3 we survey some of the existing studies on the dynamics of the web. Our focus is on the growth of the web and the update dynamics of individual documents on the web. In Section 4 we provide an overview of the FAST Crawler and describe how its design meets the challenges of web growth and update dynamics. We continue in Section 5 with a similar description of the indexing and search engines. Finally, we outline some future challenges and provide some benchmarking figures in Section 6 and Section 7, respectively.

The FAST Search Engine technology is used as a case study throughout the paper. The focus of the paper is on how web dynamics pose key challenges to large-scale Internet search engines and how these challenges can be addressed in a practical, working system. The main contribution of this paper is to offer some insight into how a large-scale, commercially operated Internet search engine is actually designed and implemented.

2 A Search Engine Reference Model

Most practical and commercially operated Internet search engines are based on a centralized architecture that relies on a set of key components, namely Crawler, Indexer and Searcher. This architecture can be seen in systems including WWW [McB94], Google [BP98], and our own FAST Search Engine.

- **Definition: Crawler**. A crawler is a module aggregating data from the World Wide Web in order to make them searchable. Several heuristics and algorithms exists for crawling, most of them are based upon following links.
- **Definition: Indexer**. A module that takes a collection of documents or data and builds a searchable index from them. Common practices are inverted files, vector spaces, suffix structures and hybrids of these.
- **Definition**: **Searcher**. The searcher is working on the output files from the indexer. The searcher accepts user queries, runs them over the index, and returns computed search results to issuer.

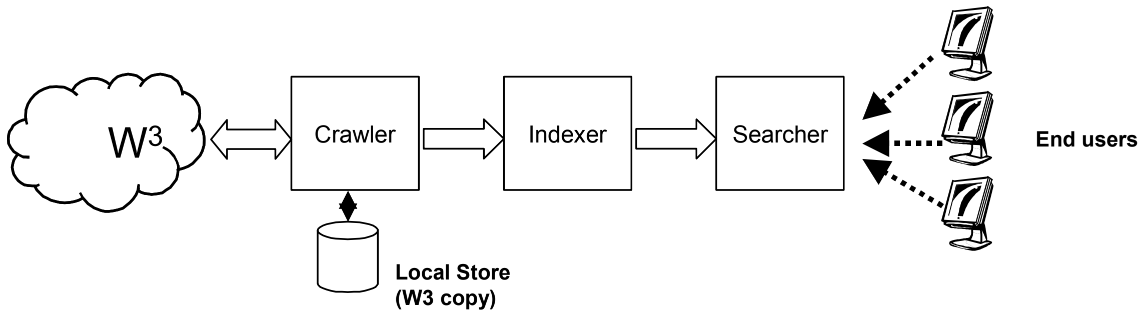

Figure 1 Search engine reference model.

The overall model is illustrated in Figure 1.

Some systems also keep a local store copy of the crawled data.

- **Definition: Local Store**. A local store copy is a snapshot of the web at the given crawling time for each document.

Systems usually run the crawler, indexer, and searcher sequentially in cycles. First the crawler retrieves the content, then the indexer generates the searchable index, and finally, the searcher provides functionality for searching the indexed data. To refresh the search engine, this *indexing cycle* is run again.

In real systems, the different phases of the indexing cycle may overlap, be split into different sub-phases, and so on. For instance, it is unacceptable to stop the searcher while running the crawler or indexer. However, in most search systems these fundamental steps of the indexing cycle are easily discernible.

The literature often distinguishes between *batch* and *incremental* crawlers. (Different papers use different terms and definitions. This paper uses the definitions by Cho and Garcia-Molina [CGM00b].) A batch crawler starts from scratch with an empty local store in every indexing cycle, and it never fetches the same document twice in that cycle. An incremental crawler never erases its local store. When it has retrieved a sufficient number of documents, it continues by re-fetching documents and updating the local store copies. When a new indexing cycle starts, an incremental crawler continues where it left off when the last indexing phase started.

3 The Dynamics of the Web

In this section we outline the nature of web dynamics. We define the different aspects of web dynamics, and we review the literature on the topic. We do not attempt to provide a complete review of the published studies but rather focus on a number of representative and significant works.

3.1 Dimensions of Web Dynamics

The concept of web dynamics has several dimensions and all are important for large-scale Internet search engines. First, the web grows in size, and most studies agree that it grows at an exponential rate. This poses a serious challenge of scalability for search engines that aspire to cover a large part of the web.

Second, the content of existing web documents is updated. A search engine must not only have good coverage, but perhaps even more important, the local store copy must be fresh. A search engine must not only scale to a large volume of documents, but it must also be able to refresh these documents in a timely manner. This requirement will only grow more important as people start turning to search engines for breaking news and other dynamic content.

Furthermore, the link structure of the web is in constant flux. Links between documents are constantly being established and removed. The dynamics of the link structure is important for search engines as long as link structure is an important component in ranking of search results. It is likely that using link structure in ranking creates a slow-working positive feedback loop in the entire web to make popular sites even more popular at the expense of less known or new sites.

A number of studies of link structure have been made, for instance the recent study by Broder *et al* [Bro+00]. However, none of these studies cover the dynamics of the link structure. We do not consider dynamics of the link structure in this paper.

Yet another dimension of web dynamics is introduced with structure. XML is evolving rapidly in inter-service systems, and the ability for a search engine to get under the hood of a presentation engine to understand the structure and semantics of the data is a key feature for the next generation engines.

3.2 Web Growth Dynamics

The web has grown at an incredible rate since the very inception about ten years ago. The web started out with a one-to-one connection between the data to be made available and the HTML page used to present the information. The web has been the driving factor for establishing a new era for storage and retrieval of information, and we are seeing the beginning of yet another. The web took storage from the application era into the information era, and the web is actually a huge storage and retrieval network.

HTML is a format description for documents. The combination of URIs and HTML made a connection between a file system object and an URI obvious, and web servers are today most commonly applications that serve HTML files directly from a file system upon requests. However, the enormous growth in information that we want to publish on the web has created the need and space for more advanced publication systems tying business applications to web servers.

The web being the platform for a new information era has caused the web to evolve into an application and transactional space as well. E-trading is becoming a major application on the web.

These two evolutionary trends on the web have erased the connection between HTML documents, URIs, and the actual content being presented. This has furthermore limited the percentage of the web that is actually indexable by a search engine. E-trading and advanced information systems introduce personalized or transaction-dependent content that is not normally accessible by standard web aggregation methods.

Several studies of the size of the web have been conducted. OpenText conducted a very early study in late 1995 [OT96]. It showed about 223,000 unique web servers, and the number of documents estimated to be 11.4 million.

In 1998 and 1999, Lee Giles and Steve Lawrence conducted well-known tests to estimate the size of the web [LG98], and to explore the accessibility of the indexable web [LG99].

In [LG98] the study is based upon multivariate analysis of the search engine coverage and overlap to estimate the size of the web. Six search engines were used, and the lower bound estimate for the size of the web was 320 million pages. Since the study is based on search engines and the result sets overlap, the measure is clearly on the indexable web, not taking into account the percentage of the web that is not touched by any search engine. Looking at the estimated coverage of each search engine with respect to the combined coverage, HotBot was the most comprehensive at the time of that measure (approximately 57% coverage).

The test was repeated and extended in Nature in 1999 [LG99]. The 11 months that had passed showed a significant increase in the number of indexable documents found. The lower bound of size was estimated to 800 million documents, and the search engines had significant less coverage of the indexable web. The maximum coverage estimated was 16%.

In [HA99], a theory for the growth dynamics of the World Wide Web is presented. Two stochastic classes are considered, namely the growth rates of pages per site, and the growth of new sites. A universal power law is predicted for the distribution of the number of pages per site. The paper brings theories that enable us to determine the expected number of sites of any given size without extensive crawling.

These three papers discuss the growth in what is referred to as the “indexable” web, but no study was performed of the percentage of pages “indexable” versus “non-indexable”. A study by Bright Planet LLC [BP00] introduced the concept of the “Deep Web”.

The deep web is easily identified as the subset of the web not discussed in [LG98] and [LG99], the “non-indexable” web. The percentage of web pages belonging to the non-indexable category is growing at a much higher rate than the indexable pages. This is a natural cause of the web moving from a simple document share space into an information sharing space and even into an application sharing space.

Key findings are in the study by Bright Planet are:

- 7500 terabytes of information (19 terabytes assumed to be the surface web).
- Approximately 550 billion documents.
- Largest growing category of web information.
- Highly relevant content found in the deep web.

3.3 Document Update Dynamics

A number of challenges must be faced to handle document update dynamics in a large-scale Internet search engine. First, we must develop a model for how documents are updated. Then, we must develop a crawling strategy that maximizes the freshness of the local store given this document update model. To evaluate the performance of various update strategies, we need mechanisms for measuring the freshness of the local store.

These challenges have been studied in the existing literature. Cho and Garcia-Molina have published several studies both on how web documents are updated and on crawling strategies [CGM00, CGM00b, CGM00c]. Their models and experiments indicate that web document updates can be modeled as independent Poisson processes. That is, each document d_i is updated according to a Poisson process with change rate λ_i, and the change rates are independent. Their experiments on the web indicate that an average document is changed once every ten days and that 50% of all documents are changed after 50 days [CGM00b].

Cho and Garcia-Molina have also developed statistical estimators for the Poisson parameters under various assumptions. They have derived estimators for uniform and random observation of the web documents, and for known and unknown time of last document update. The new estimators are much better than the naive estimator: the number of document updates observed divided by the observation time [CGM00c].

Finally, they have also presented an optimal crawling strategy given their model of a crawler and local store. They also propose a framework for measuring how up-to-date the local store is through their concepts of *freshness* and *age*. They define the freshness of a document d_i at time t as the probability that the document is up-to-date at the given time. Age is defined to be the 0 for documents that are up-to-date and the time since the last document update in the real world for others. For both freshness and age, the interesting measure is the average freshness or age both over all documents and over time [CGM00]. In this paper we will use the term "freshness" in a more informal manner.

Based on their models for document change and freshness measures, they present some optimal scheduling policies for an incremental crawler. They make the following observations:

- Refreshing document using uniform update frequencies is always better than using document update frequencies that are proportional to the estimated document change frequencies λ_i.
- The scheduling policy optimizing freshness penalizes documents that are changed too often. Intuitively, these documents are likely to change again very soon and hence do not contribute much to the overall freshness of the local store.
- The scheduling policy optimizing age favors documents that are changed very often, but the actual change is small.

Brewington and Cybenko [BC00] also performed a number of experiments to discover how web documents are updated. They also conclude that web documents are updated according to a Poisson process. Brewington and Cybenko also propose a novel measure of freshness, termed (α,β)-currency. A document d_i is (α,β)-current if it were up-to-date β time units ago with probability α. This measure captures both the *aspirations* and the actual *achievements* regarding freshness. A daily newspaper may be (0,95, 1 day)-current, meaning that 95% of all articles in the paper was up-to-date one day ago. They estimate that an Internet search engine containing 800 million documents must refresh 45 million documents every day to be able to maintain (0,95, 1 week)-currency.

Edwards, McCurley, and Tomlin [EMT01] present a crawler that minimizes the number of obsolete documents in the repository *without* making any a priori assumptions about how documents are updated. They use measured document change rates and divide their crawling resources on documents according to their change rate. They solved a vast optimization problem to find the optimal distribution of crawling resources.

3.4 Search Engines and Search Technology

There is not a rich body of literature describing practical large-scale crawling and searching systems, and in particular, very few address issues relating to the dynamic web. Brin and

Page have described an early version of the Google system [BP98]. They address issues relating to growth of the web and scaling of the document volume, but they do not address refreshing of the local store.

Heydon and Najork describe their scalable web crawler [HN99]. Their crawler is scalable in the sense that the required machine resources are bounded and do not depend on the number of retrieved documents. Their crawler reportedly runs on a single, large machine, but it can probably quite easily be implemented in a crawler cluster to scale with even larger document volumes or processing load in a manner similar to our FAST Crawler as discussed later in this paper. Heydon and Najork do not discuss refreshing the crawled documents.

Edwards, McCurley, and Tomlin [EMT01] provide some design details about their crawler. This crawler runs *incrementally*, constantly refreshing documents to ensure freshness over time, as described in the previous section. They also provide some details about their scheduling algorithm. Each document is classified according to its measured update frequency, and crawling resources are then divided among these classes. They formulated and solved a vast optimization problem for computing how to allocate the crawling resources among the different document update classes.

4 Aggregation of Dynamic Content

In this section, we use the FAST Crawler as a case study to illustrate how we have addressed the challenges of scaling with the size of the web and ensuring the freshness of our local store.

4.1 Overview of the FAST Crawler

The FAST Crawler consists of a cluster of interconnected machines as depicted in Figure 2. Each machine in this cluster is assigned a partition of web space for crawling. All crawler machines communicate with all other machines in a star network. However, the machines work relatively independent of each other, only exchanging information about discovered hyperlinks.

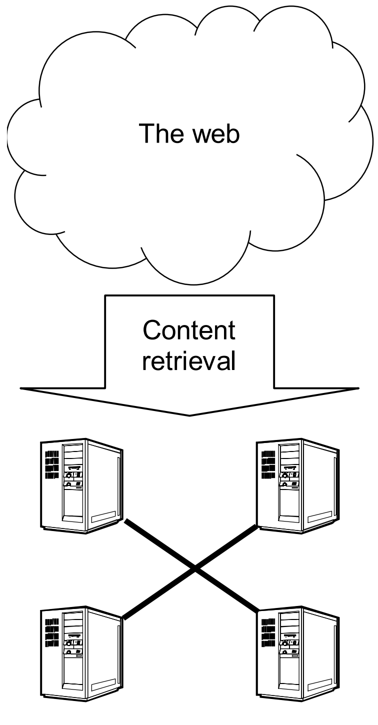

Figure 2 FAST Crawler deployment overview.

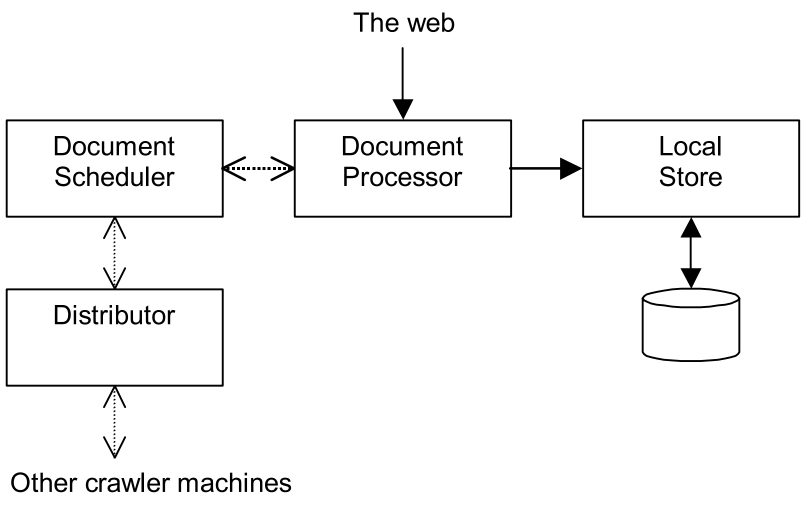

Figure 3 Components and data flow in a single crawler machine.

Figure 3 depicts the main components of a single crawler machine in our system. The solid arrows indicate flow of document content, and the dotted arrows indicate flow of document URIs, possibly with associated meta-information.

The *Document Scheduler* is responsible for figuring out which document to crawl next. Conceptually, this means maintaining a huge prioritized queue of URIs to be crawled while also observing the informal “social code” of web robots such as the robots exclusion protocol [Kos94] and various constraints on the access pattern to individual web servers imposed to avoid overloading servers with crawler requests. The Document Scheduler sends a stream of URIs to the Document Processor.

The *Document Processor* is responsible for retrieving documents from web servers and performing any processing on them. This component consists of a *manager* module that provides a plug-in interface used by a set of *processing* modules. The manager routes documents through a pipeline of processing modules based on configuration information, the document content type, etc. This architecture provides the necessary flexibility to quickly support new document processing functionality in the crawler. We currently have processing modules for parsing HTML documents and various multimedia formats, classification of language, classification of offensive content, etc. It is the HTML parser that discovers hyperlinks to other documents and passes these back to the scheduler.

After processing, the crawler stores relevant document content and meta-information in the *Local Store*. Document content represents the bulk of the data, and we have optimized our storage system for efficiently writing or updating individual documents and for streaming the entire document collection to the indexing step. There is no need for efficient random access reading of document content. We use a hybrid of a hashing and logging storage system. Hirai *et al* describe the WebBase system, a system that is very similar to our storage system [HRGP00]. Some document meta-information is kept in a high-performance, random access database.

The *Distributor* exchanges hyperlink information with other machines in the crawler cluster. There is a static mapping from the relevant hyperlink information to a crawler machine.

The crawler architecture also contains two important components for duplicate detection and link-based ranking, respectively. These modules are relatively complex and outside the scope of this paper.

The FAST Crawler is an *incremental crawler*. That is, when the document repository has reached its target size, the crawler continues by refreshing existing documents to detect changes. It fetches new documents only to replace documents that are deleted from the repository because the crawler discovers that they have been deleted from the web or for other reasons. The crawler is never stopped. It is temporarily suspended during a part of the indexing process, but when indexing is completed the crawler resumes operation where it left off.

4.2 Scalability

A large-scale crawler must be scalable both with respect to *document storage capacity* and *document retrieval capacity*. In our architecture, each crawler machine is responsible for all retrieval, processing, and storage for a partition of the document space. The different machines constituting a crawler cluster work independently except for exchanging discovered URIs. Hence, the storage capacity, C_S, and the processing capacity, C_P, of the crawler cluster is the sum of the capacity of each individual machine:

$$C_S = \sum_i C_{S,i}$$

$$C_P = \sum_i C_{P,i}$$

An additional constraint on the crawler retrieval and processing capacity is the capacity defined by the total network bandwidth available to the crawler cluster, C_N. Hence, the total retrieval capacity of the crawler cluster is

$$C_R = \mathrm{Min}(C_P, C_N).$$

Usually, network bandwidth represents the highest cost for running a large-scale crawler and hence it makes sense to dimension the system so that $C_N \leq C_P$. Note that the bandwidth used for communicating internally in the crawler cluster is proportional to the inbound bandwidth used for retrieving content from the web. The internal bandwidth is only used for exchanging hyperlink information, and the number of hyperlinks is proportional to the number of retrieved documents.

We can easily increase the document storage capacity, C_S, by adding new machines to the crawler cluster and redefine the workload partitioning in the distributor accordingly. This will also give us some extra processing power, C_P, "for free". Increasing only the document storage capacity does not require more network bandwidth, C_N, between the crawler and the web or internally between the individual crawler nodes comprising the cluster. As a result, the crawler scales linearly with document storage capacity.

We also scale linearly with document retrieval capacity, C_R. Retrieving, processing, and storing more documents per unit of time requires a linear increase in inbound network bandwidth from the web and also a linear increase in network bandwidth between the machines comprising the crawler cluster, C_N.

Scaling document retrieval capacity by increasing network capacity assumes that each machine in the crawler cluster has enough spare processing capacity to handle the increased load, that is $C_N \leq C_P$. If this is not the case, then additional machines must be added to the cluster to achieve the required processing power. This will also increase the document storage capacity. Hence, the total system scales linearly with both storage and retrieval capacity as long as there is a reasonable balance between the two. In practice, this is usually the case, as we do not want the ratio of retrieved documents per unit of time to the total number of documents to drop as we scale the system, as this will eventually impact our freshness. For a crawler to maintain a given freshness, F, the following ratio must be kept constant while scaling (let k be any constant):

$$k \frac{C_R}{C_S} = F$$

A system that scales to many components must also be robust with regards to failure of any of these components or otherwise the probability of having a working system will drop with the number of components. Our crawler is robust against failure of any machine in the crawler cluster. Each machine works independently on scheduling, retrieving, processing, and storing documents. The only dependency between machines is the exchange of information about new hyperlinks. Hyperlink information for an unavailable crawler machine is simply queued on the sending machine until the designated receiver again becomes available.

4.3 Freshness Through Scheduling

The FAST Crawler provides fresh data to the search engine through its very high document retrieval capacity and its scheduling algorithm prioritizing retrieval of documents most likely to have been updated on the web. A good scheduling algorithm increases the constant factor *k* in the freshness equation in the previous section.

In normal operation, the crawler is refreshing a local store of approximately a constant number of documents limited by the capacity of the search engine and the crawler itself. In this state, the crawler will only retrieve new documents when old documents are removed from the local store either because the crawler attempted a refresh but the document does not exist on the web anymore or for other reasons. When the document storage capacity is increased, we normally relatively quickly retrieve enough documents to again reach this steady state.

Providing fresh search results to users implies short indexing cycles, and we do not have the capacity to refresh all documents in our local store between each indexing cycle. In this situation the scheduling algorithm is a key element in ensuring a fresh local store. To maximize freshness, we must spend as much as possible of our crawler network capacity on refreshing documents that have actually changed.

The FAST Crawler currently uses a relatively simple algorithm for adaptively computing an estimate of the refresh frequency for a given document. Basically, this algorithm decreases the refresh interval if the document was changed between two retrievals and increases it if the document has not changed. This is used as input to the scheduler, which prioritizes between the different documents and decides when to refresh each document. In addition, the scheduler is configured with global minimum and maximum refresh intervals to keep the refresh rate for a document within sensible bounds, e.g. to allow refreshing of documents for which we have never observed a change.

Cho and Garcia-Molina observe that it is not always optimal to refresh documents that are updated very frequently “as often as possible” [CGM00, CGM00c]. Intuitively, these documents will always be obsolete anyway when the repository is indexed. In fact, they conclude that a uniform refresh policy where all documents are updated equally often is *always* superior to such a *proportional* refresh policy. In practice, this is not a big problem. With a repository of the size required for a large-scale search engine, there are always enough documents waiting to be refreshed to diminish the effect of the relatively few documents that are updated very often. Also, we configure the scheduler to avoid rescheduling any document more than once for each indexing cycle.

4.4 Freshness Through Heterogeneity

When optimizing freshness through scheduling, we assume that all documents have the same freshness requirements. Having a fresh copy of one document in the repository is just as valuable as having a fresh copy of any other document. This is not always the situation in the real world. Consider the following two examples:

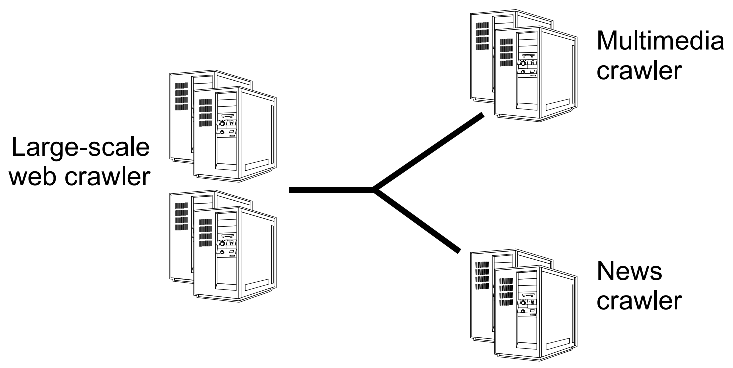

Figure 4 Example of heterogeneous crawler deployment with sub-clusters for different crawling strategies.

- There is an industry trend to offer a “pay for inclusion” service to content providers. This offer usually comes with service level guarantees covering refresh and indexing intervals.
- There is a trend towards searching in increasingly dynamic content, e.g. news. This content must be refreshed and indexed very often to be of any value to users.

The above two scenarios cannot be supported in a large-scale crawler using a relatively simple scheduling algorithm alone. In the FAST Crawler, we have solved this problem by permitting heterogeneity in the cluster of crawler machines. This means that the different machines can be configured with very different storage and processing capacity, $C_{S,i}$ and $C_{P,i}$. A relatively small number of machines are dedicated to special purposes, such as crawling of content from paid inclusion programs or news sites. These machines can be configured specially with the service requirements of these services in mind, and we can control the load on these machines without sacrificing the high capacity and efficiency of the main bulk of the machines in the crawler cluster.

We still keep these dedicated machines as a part of the cluster just as all the other machines so that all machines can efficiently share link information, computed link ranks, etc. The cost is relatively small — only a small increase in the complexity of the distributor.

Figure 4 shows an example of a heterogeneous crawler cluster. This figure depicts a crawler cluster with a few dedicated machines for crawling news content in addition to the large-scale web crawler. We have also included a few machines comprising a multimedia crawler to illustrate the flexibility of the architecture. In this example, the large-scale web crawler will typically be configured with a large C_S but a relatively low F. The news crawler will be configured with a lower C_S to maintain a high F. The multimedia crawler will have high C_S and low F just as the large-scale web crawler, but operating this crawler as a separate part of the cluster allows control over resources used for different media types.

4.5 Cooperation with Providers

Another approach, that we are experimenting with, is cooperating with content providers to further improve freshness. There are different models for cooperating with content providers, and in this section we outline the different options.

InfoSeek once proposed a standard for a web server meta-file named *siteinfo.txt* to

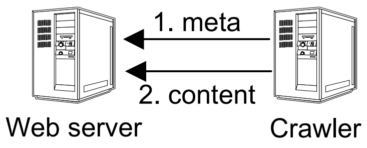

Figure 5 Meta-information pull model.

complement the more established *robots.txt* de facto standard. The purpose of the *siteinfo.txt* file was to provide information to crawlers about which documents had changed, mirrors of the server, etc. The *siteinfo.txt* standard was never able to establish itself and our own crawling indicates that today no server is using it. Today, a number of content providers use proprietary meta-files to publish information about their sites and how they are updated.

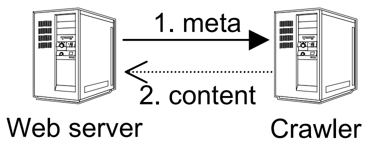

Figure 6 Meta-information push model.

This is a *pull* model that permits the crawler to perform efficient per-site scheduling. In this model, the crawler periodically fetches some meta-information from the web server. It uses this information to influence its own scheduler with hints about new or changed documents. This model is illustrated in Figure 5. (The arrowheads indicate the direction of the requests.) However, building a crawler that supports all these proprietary meta-information formats is a daunting task.

A more elaborate approach is to *push* content or meta-information directly from the content

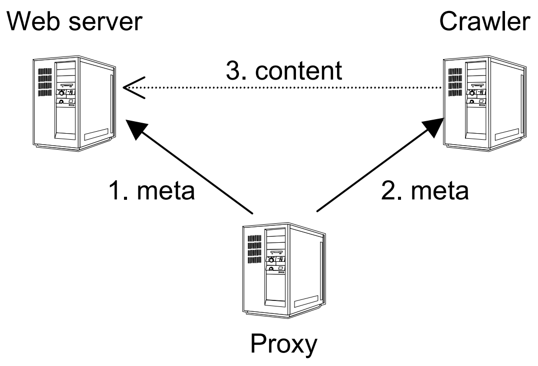

Figure 7 Hybrid meta-information pull-push model.

provider to the crawler. In this model the content provider sends a notification to the crawler whenever a document is added, modified, or deleted on a site. In its simplest form, the content provider just submits the URI of the document. This method can be enhanced by also submitting various meta-information, or even the entire content of the documents. However, care must be taken to avoid opening the crawler and search systems to spamming and other abuse so such a service can only be offered to partners. This model is illustrated in Figure 6 with the dotted arrow indicating an optional request.

A third approach is a hybrid between the two above. There are many obstacles to having push technology deployed at a large set of content providers. For instance, site operators may be reluctant to increase the complexity of their systems by installing additional software components. The hybrid approach involves developing and deploying special *proxy systems* that read site meta-information files, act according to the instructions in those files, and then use push technology to update the crawler. These systems use a single interface to the crawler but are otherwise independent of the crawler, and hence these proxies can be developed and deployed independently. This is illustrated in Figure 7.

The hybrid model is quite powerful. The proxy may not limit itself to obtaining meta-information from only the server or set of servers being crawled. For instance, it can also fetch meta-information from other sources to discover “hot spots” in the web and then direct the crawlers to these spots.

After establishing the communication pipeline with the providers, a natural next step lies within better understanding the content of the provider. XML could be a significant player for describing both semantics and dynamics of the content.

5 Searching Dynamic Content

The second and third components in the reference search engine model, the Indexer and the Searcher, need also to handle the different dimensions of web dynamics. Traditionally, search engines have been based upon batch-oriented processes to update and build indices. To handle the growth in size of the Web and the update dynamics, most traditional designs fall short. In this section, we will study several aspects and solutions for an indexer and a searcher to handle a dynamic web.

5.1 Scalability with Size and Traffic

Being able to handle the web growth calls for architectural solutions. Given traditional solutions with inverted files or equivalent, the cost of building, maintaining and searching an index is worse than linear.

One possible architecture for a Web search engine is the FAST Search Engine Architecture. It handles scalability in two dimensions, namely size and traffic volume. The architecture is a distributed architecture, and has two classes of nodes:

- Search Nodes: A Search node S_i holds a portion of the index, I_i. The total index is

  $$I = \bigcup_i I_i$$

  Each of the search nodes is a separate entity than holds a search kernel (searcher) that searches the index I_i and returns search results. The search nodes have no interconnection between them.
- Dispatch Nodes: A dispatch node does not hold any searchable data. The dispatcher is a routing, distribution, and collection/merging mechanism. A dispatch node receives queries and routes them to a set of underlying search nodes, $S_i..S_j$. The results are collected and merged before they are sent to the issuing client.

A search node has two capacities, namely the number of documents on each node, $|D_i|$, and the query rate or traffic capacity, C_i. A dispatcher has one capacity, namely the dispatching capacity, C_{di}. The C_{di} depends on the number of search nodes the query is sent to.

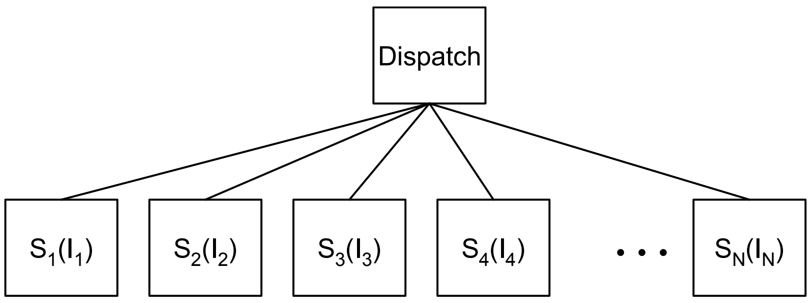

Figure 8 Linear Scaling with data size

Now, by using the two components described above, we can build a simple architecture to allow linear scaling with the data size. The architecture is shown in Figure 8. Each box $S_i(I_i)$, is a search node holding the index partition I_i. The entire dataset, I, is partitioned out on the

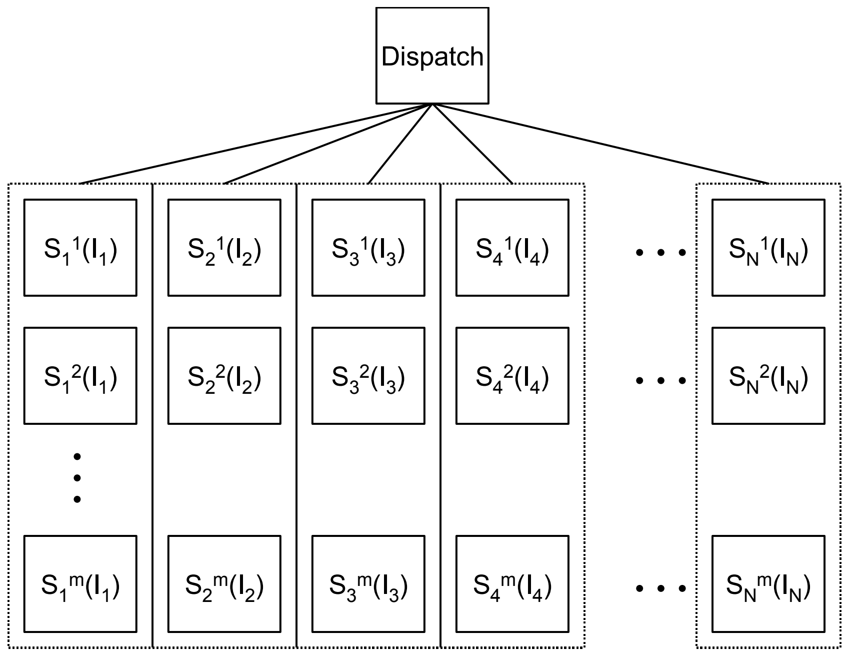

Figure 9 Scaling in size and capacity

search nodes. The dispatch node is broadcasting queries to all search nodes in parallel and merging results from the search nodes to build the final result set.

Now, assuming that each search node handles the query individually of the other search nodes, we see that a linear scale up for increasing the overall size $|D|$ is achieved.

At the same time, to scale with the query rate, replication will provide the capacity $m * C_i$. The dispatch node will know all search nodes in a column, and a round-robin algorithm can be used to rotate between the different columns. This extended architecture is illustrated in Figure 9. Here S_{ij} is a search node j handling partition i. As illustrated, the dispatcher knows all search nodes in each column, and can load-balance between them, achieving a linear performance scale up.

So, the number of search nodes required for a dataset I of size $|I|$ is derived as $|I|/p$, where p denotes the optimal size of a search node partition. This number is of course dependent on actual hardware configurations and costs. Furthermore, the number of search nodes to handle a given query rate Q_t, is derived as Q_t/C_i.

The limitation of this architecture clearly lies within the dispatching system. A dispatcher has a capacity of merging results. Optimal merging algorithms are in order of $O(r \log m)$, where r denotes the number of sources and m the number of entries. Thus, the merge performance is limited by the number of search nodes that receive the query in parallel.

To ensure linear scalability, a multi-level dispatch system can be configured. By letting a dispatcher dispatch to a part of each row, and the letting a super-dispatcher dispatch to those dispatch nodes, we can use a tree-like architecture, as shown in Figure 10.

Any number of levels can be built to accommodate for the scale in the two dimensions. In the worst case, this will be a binary tree, where the number of nodes is $O(2N-1)$, and still linear to the size of the data being searched.

An immediate observation from the description above is that the architecture also has implicit fault-tolerance. By having multiple nodes with the same index partition, I_i, dispatchers can be fault-tolerant by detecting timeouts or non-replies. To ensure fault-tolerance on any level, the transparency of dispatcher/searcher can be utilized to have redundant dispatcher.

5.2 Handling Update Dynamics

The second dimension of web dynamics, the update dynamics, is also a problem facing search engines. Traditionally, structures used for indexing are based upon offline building, and in many cases this is a highly time consuming process.

There are two ways to make indexing processes more dynamic:

1. Identifying new inverse structures that allow for online updates.
2. Utilizing a heterogeneous architecture to allow for dynamic changes to the dataset.

For 1) there are several proposed solutions in the IR community, but none that has gotten big acceptance in the industry.

The FAST Search engine uses the second approach to cope with dynamic updates. Looking at the distributed architecture described above, the indexing is an easy goal for parallelization. Indexing D can be done by indexing I_1, I_2, … I_N individually. Since separate search nodes handle each partition, indexing can be done individually.

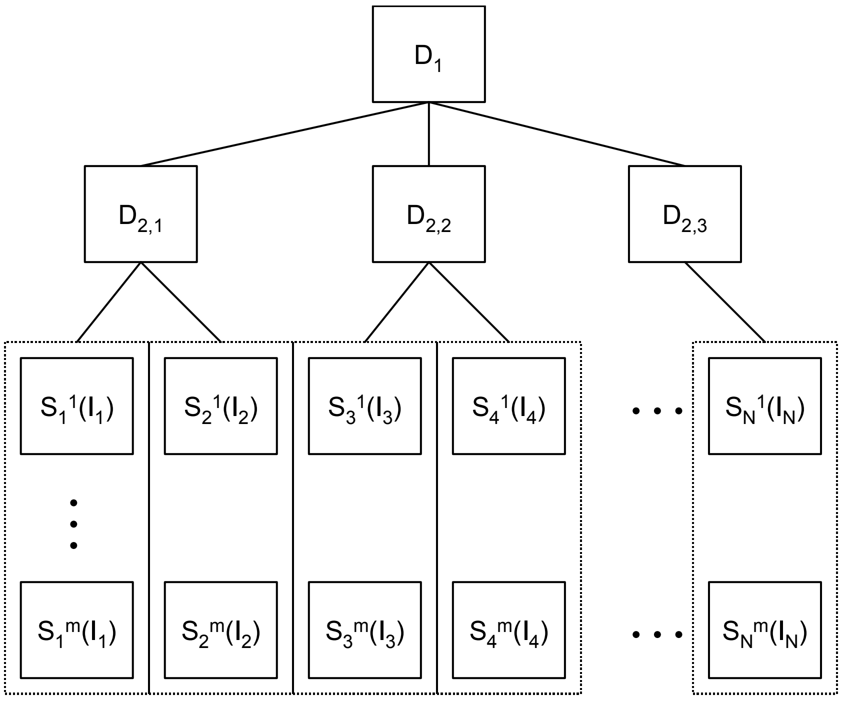

Figure 10 Multi-level dispatch architecture

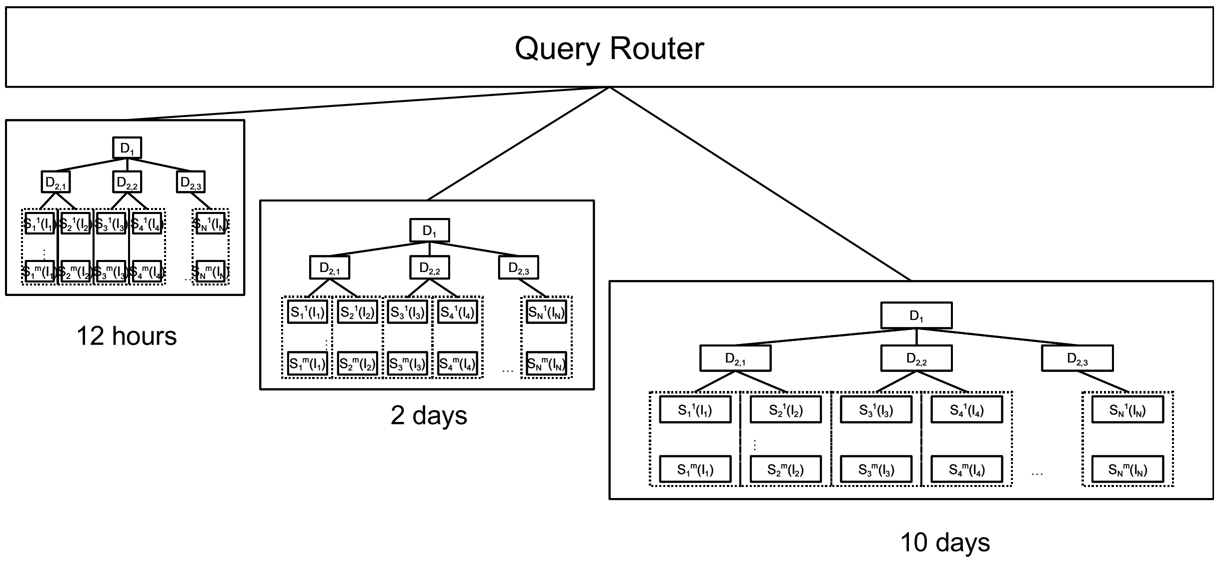

Figure 11 Multi-tier cluster.

This solution still does not make indexing possible online on each search node, but by having slight overcapacity of search nodes, we can easily switch nodes on and off line to update parts of the dataset. Still, the system is batch-oriented and the time from detecting a document update until it is pushed live will still be 20 hours or above.

By using a crawling system that allows heterogeneous clusters of crawler nodes, it is easy to identify sources with different update rates. Taking the architecture described in 5.1 and using that as a building block, we can have a cluster of search clusters where each cluster has a different update frequency.

A possible cluster solution is shown in Figure 11. In this example, we deploy three different clusters, each receiving a data feed from a different crawler cluster. The update frequencies of the clusters can then be different, but in general update cost (in either time or required hardware) is linear to the data size.

6 Future Challenges

The evolution of the dynamic web raises several significant challenges for search engines. First, the increasing dynamics and size makes intelligent scheduling increasingly important. Being able to update the most important parts of an index at a timely rate will be crucial in order to bring relevant search results to the users. Intelligent scheduling, heterogeneous crawling and push technology will be crucial to building aggregation and search systems capable of scaling with the web at a reasonable price.

The size of the web is clearly a big challenge, and one important question is arising: Do we actually need to search several billion documents for every query? Being able to intelligently classify and predict a probable subset of the data set to search for given queries will enable us to build much more efficient and cost-effective solutions.

At the same time, the “Deep Web” is most likely growing at a rate much higher than the current “indexable” web. There is no unified and practical solution to aggregate the deep web on a large scale, but push based technology and perhaps tight integration into publication and content management systems will evolve to address this challenge.

The explosive growth of the web also calls for more specific search engines. The introduction of focused crawling and document classification enables both crawlers and search engines to operate very efficiently within a topically limited document space. The Scientific search engine **scirus.com** is a good example of an engine that uses both focused crawling along with document classification. The dynamics also has a more homogenous character within such a vertical, enabling a fresher search experience.

7 Conclusion

We have discussed several dimensions of web dynamics. Both growth and update dynamics clearly represent big challenges for search engines. We have shown how the problems arise in all components of a reference search engine model.

The FAST Search Engine architecture copes with several of these problems by its key properties. The overall architecture that we have described in this paper is quite simple and does not represent very novel ideas. The system architecture is relatively simple, and this makes it manageable even when it grows. In a real-life system with service level requirements, simplicity is crucial to operating the system and to being able to develop it further.

Being heterogeneous and containing intelligence with regards to scheduling and query processing makes this a real-life example of dealing with web dynamics issues today. The service running at **www.alltheweb.com** and major portals worldwide currently handle more than 30 million queries per day. Indexing happens every 11 days, and the full index size is currently about 700 million full-text documents. These documents were selected from a crawled base of 1.8 billion full-text documents. The crawler architecture enables us to crawl at rate of 400 documents/second and beyond.

The system is based on inexpensive off-the-shelf PCs running FreeBSD and our custom search software. We currently use approximately 500 PCs for our production systems. Most of these machines are search nodes. We currently use 32 machines for crawling. The hardware configuration differs depending on the role the machine has in the architecture and the time of acquisition. Most machines are typically dual-Pentium machines with between 512 and 1024 Mbytes of memory.

Future evolution of web dynamics raises clear needs for even more intelligent scheduling to aggregate web content as well as technology for push-based aggregation. By doing more intelligent query analysis and processing, we will be able to do a sub-linear scaling with the growth of the web based on the ideas from Figure 11. It is possible to create a multi-tier system where one tier with few columns and many rows handles a relatively large part of the most popular queries. Another tier with more columns but fewer rows can then handle the remaining queries.

References

[OT96] Tim Bray. Measuring the Web, In *Proceedings of the Fifth International World Wide Web Conference (WWW5),* 1996.

[BC00] Brian E. Brewington and George Cybenko. How Dynamic is the Web? In *Proceedings of the Ninth International World Wide Web Conference (WWW9).* 2000.

[BP98] Sergey Brin and Larry Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh International World Wide Web Conference (WWW7).* 1998.

[BP00] The Deep Web: Surfacing Hidden Value. White Paper, Bright Planet 2000.

[Bro+00] Andrei Broder *et al.* Graph Structure in the Web. In *Proceedings of the Ninth International World Wide Web Conference (WWW9).* 2000.

[CGM00] Junghoo Cho and Hector Garcia-Molina. Synchronizing a Database to Improve Freshness. In *Proceedings of 2000 ACM International Conference on Management of Data (SIGMOD).* 2000.

[CGM00b] Junghoo Cho and Hector Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler. In *Proceedings of the 26th International Conference on Very Large Databases.* 2000.

[CGM00c] Junghoo Cho and Hector Garcia-Molina. Estimating Frequency of Change. Unpublished. 2000.

[EMT01] Jenny Edwards, Kevin McCurley, and John Tomlin. An Adaptive Model for Optimizing Performance of an Incremental Web Crawler. In *Proceedings of the Tenth International World Wide Web Conference (WWW10).* 2001.

[HN99] Allan Heydon and Marc Najork. *Mercator: A Scalable, Extensible Web Crawler.* World Wide Web. Vol 2(4). December 1999.

[HRGP00] Jun Hirai, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. WebBase: A Repository of Web Pages. In *Proceedings of the Ninth International World Wide Web Conference (WWW9).* 2000.

[HA99] Bernardo A. Huberman and Lada A. Adamic. Evolutionary Dynamics of the World Wide Web. Technical Report. Xerox Palo Alto Research Center. February 1999.

[Kos94] Martijn Koster. *A Standard for Robots Exclusion.* 1994. http://www.robotstxt.org/wc/robots.html.

[LG98] Steve Lawrence and C. Lee Giles. *Searching the World Wide Web*. Science. April 1998.

[LG99] Steve Lawrence and Lee Giles. *Accessibility and Distribution of Information on the Web*, Nature. July 1999.

[McB94] Oliver A. McBryan. GENVL and WWWW: Tools for Taming the Web. In *Proceedings of the First International World Wide Web Conference (WWW1).* 1994.

[SRR97] Richard Seltzer, Eric J. Ray and Deborah S. Ray. *The AltaVista Search Revolution.* Osborne McGraw-Hill. 1997.