Trovatore: Towards a Highly Scalable Distributed Web Crawler

P. Boldi, S. Vigna
Dipartimento di Scienze dell'Informazione
Via Comelico, 39/41
I-20135 Milano, Italy
{boldi,vigna}@dsi.unimi.it

B. Codenotti
Istituto di Matematica Computazionale – CNR
Via Moruzzi, 1
I-56124 Pisa, Italy
codenotti@imc.pi.cnr.it

M. Santini
Istituto per le Applicazioni Telematiche – CNR
Via Moruzzi, 1
I-56124 Pisa, Italy
santini@imc.pi.cnr.it

ABSTRACT

Trovatore is an ongoing project aimed at realizing an efficient distributed and highly scalable web crawler. This poster illustrates the main ideas behind its design.

Keywords

Distributed crawling, web searching, fault-tolerance, self-stabilization, delegation function, Java, RMI.

1. INTRODUCTION

The main goal of this project is the design and implementation of a distributed highly scalable web crawler. As a consequence, essential features of the software being developed are platform-independence, and tolerance to transient failures. In addition, the software will make it possible to recover from permanent non-destructive failures in an easy manner. The needs of efficiency and fault-tolerance imply seeking trade-offs between communication complexity and data replication.

An essential property of our system is self-stabilization (in the classical sense defined by Dijkstra [3]). Although guaranteeing this property required a significant amount of effort in the initial design phase and in the construction of the interaction protocol, it has been instrumental to achieve that agents in the system (and *a fortiori* new hardware or bandwidth resources) can be added, or obsolete parts can be removed, without interfering too much with its behaviour.

Indeed, many Internet services are actually self-stabilizing, even if no theoretical machinery or guidance was used in their design, the most notable example being the Domain Name System. Although self-stabilization is clearly unacceptable when stringent safety property are imposed on the system, we believe that the nature of the web (see Section 3) makes a web crawling system an ideal testbed for design principles based on self-stabilization.

2. THE SOFTWARE ARCHITECTURE

The software architecture consists of a number of *agents*, each one delegated to deal with a specified portion of the web domain under investigation.

The more relevant components of each agent (see Figure 1) are the following:

Store. This component deals with the storage of the crawled pages. Its main functions are: to check whether a page has already been crawled, to store the content of crawled pages, to keep track of some relevant information about its stored pages. Such information should include (but is not limited to) the out-links and in-links of each page; moreover, *Store* can also compute some statistics of its stored pages (size, time of the last crawl, . . .).

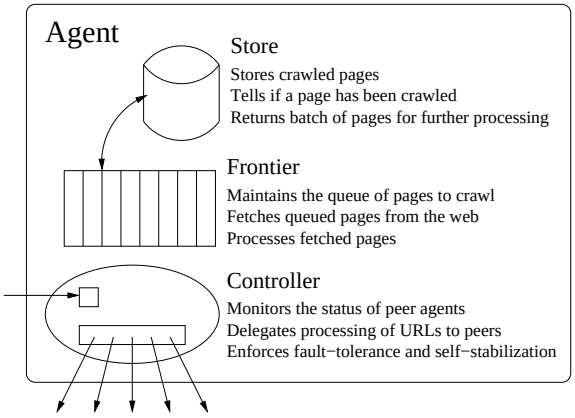

Figure 1: Internal architecture of a single agent.

Additionally, it should deal with "indexers" or other software processing its content; to this end, it should be able to return its content both in response to a single query (specified by an URL) and as a "stream", in response to a "batch" of related queries.

Frontier. This component deals with the retrieval of new pages. Its main functions are: to keep track of the URLs that have to be crawled by the agent, to actually fetch the content of the URL to be crawled and to parse the retrieved URL via different filters. In addition to accomplishing other tasks, such filters should determine the new URLs to be crawled.

Controller. It oversees all the communications between agents and works as a reliable crash-failure detector; reliability refers to the fact that a crashed agent will eventually be distrusted by every active agent (a property that is usually referred to as *strong completeness* in the theory of failure detectors).

One of the central issues in the design of the Controller is the use of a *delegation function* that determines which

agent is responsible for each single URL. The delegation function must be chosen in such a way that data replication is minimized; in particular, if some agent crashes, or deliberately decides to gracefully stop working, the URLs for which it was responsible should be partitioned among the remaining agents, thereby maintaining all other previously decided assignments.

A further requirement is that the delegation function should partition the web domain in such a way that every running agent is assigned approximately the same number of URLs.

The design of the Controller may seem to be limited essentially to some synchronization logic that delivers new URLs to be retrieved. However, this is not true if we want to design the system so that it will stabilize to a correct, balanced page distribution between agents. Indeed, this requires that the controller, before actually trying to retrieve a page, tries to check whether other agents already have this page. This can happen because the current agent was stopped for a while (so other agents were responsible for that page), or because the agent suffered a transient failure (for instance, disk full), so it previously delegated elsewhere the page retrieval. Querying all the agents is clearly out of the question. Thus, the retrieval mechanism must use the properties of the delegation function to guarantee that in the presence of a small number of faults most pages will be retrieved just once.

We are currently experimenting on some different choices for the delegation function to be used by our system.

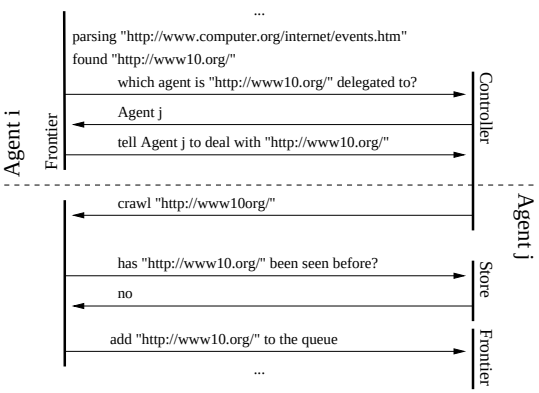

Figure 2: Inter-agent transactions.

The components of each agent interact as semi-independent modules, each running possibly more than one thread. In order to limit the amount of information exchanged on the network, each agent is confined to live in a single machine. Nonetheless, different agents may (and typically will) run on different machines, and interact using RMI (see Figure 2 where time flows downwards. Note that the figure only shows a simplified excerpt of the actual interaction).

3. IMPLEMENTATION ISSUES

We now make a little bit more precise some of our design choices. Concerning self-stabilization, the system is designed in such a way that transient faults can possibly induce a small amount of useless replication, but in a short time the system converges to the desired behaviour. We believe that for a system that interacts heavily with a complex and sometimes unforeseeable (even unfathomable) medium as the web, which however displays a globally coherent behaviour, it seems more reasonable to allow the system to temporarily behave loosely and eventually converge to a correct behaviour rather than insisting on very strict protocols whose correctness is often very difficult to prove, and that are usually not so robust in the face of unpredictable inputs.

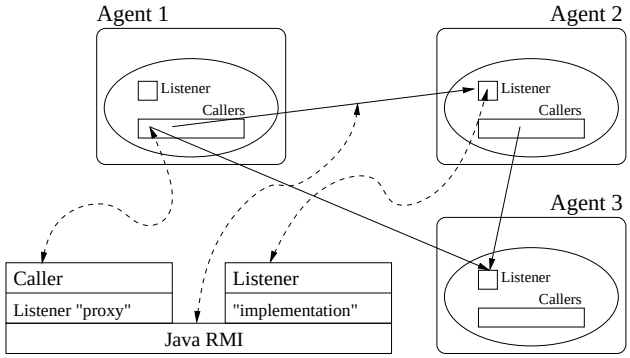

Figure 3: Some implementation issues.

In order to achieve the platform-independence goal, we have chosen Java™ 2. This is a well-established, secure, and scalable development environment equipped with many features tailored to the web. In particular, instead of explicitly implementing a dedicated network protocol for inter-agent communication, we adopt *Remote Method Invocation* [5], a technology which enables us to create distributed applications in which the methods of remote Java objects can be invoked from other Java virtual machines (possibly on different hosts), using object serialization to implicitly marshal and unmarshal parameters (see Figure 3).

4. CONCLUSIONS

We have presented the guiding ideas behind the design of Trovatore. This is an ongoing effort, and we plan to make available an on-line report on its status.

5. REFERENCES

[1] Sergey Brin and Lawrence Page. The anatomy of large-scale hypertextual web search engine. In *Proceedings of the Seventh International World Wide Web Conference*, volume 30 of *Computer Networks and ISDN Systems*, pages 107–117, April 1998.

[2] Junghoo Cho and Hector Garcia-Molina. Incremental crawler and evolution of the web. Technical Report, Department of Computer Science, Stanford University.

[3] Edsger W. Dijkstra. Self-Stabilizing Systems in Spite of Distributed Control *Communications of the ACM*, 17(11):643–644, 1974.

[4] Allan Heydon and Marc Najork. Mercator: A Scalable, Extensible Web Crawler. In *World Wide Web*, December 1999, pages 219–229.

[5] Java™ Remote Method Invocation (RMI). `http://java.sun.com/products/jdk/rmi/`