Searching Moving Objects in a Spatio-Temporal Distributed Database Servers System

Mauricio Marín[1], Andrea Rodríguez[2], Tonio Fincke[3], and Carlos Román[2]

[1] Center for Quaternary Studies, CEQUA, Universidad de Magallanes, Chile
mauricio.marin@umag.cl
[2] Universidad de Concepción, Chile
{andrea,caroman}@udec.cl
[3] University of Münster, Germany
dobrick@yahoo.com

Abstract. Querying about the time-varying locations of moving objects is particularly cumbersome in environments composed of a very large number of distributed spatio temporal database servers. In particular, searching for a specific object can require to visit each server. In this paper we propose a strategy to avoid such an exhaustive search that is based on the use of a centralized index, called *meta-index*, which is the entry point for spatio-temporal search queries. This index allows a software agent to determine a search plan for visiting the most likely servers to contain the target object. An important issue for large and dynamic distributed servers systems is to keep the meta-index as up-to-date as possible with the real system. This paper defines and compares two different strategies for maintaining properly updated the meta-index: *crawling*, where the centralized system that keeps the index controls itself the updating process, and *harvesting*, where each distributed database server autonomously transfers data directly into the central index system. Both strategies were implemented and compared by using discrete-event simulators with demanding synthetic spatio-temporal data. The results show that crawling has better performance.

1 Introduction

Systems with dynamic data often need to collect and store data in multiple disperse servers, while handling the whole or partial data access from a centralized location. Centralized access strategies maintain copies or summary of data at a central location using *replicas* [13] [11] [9] or *indexes* [8].

This paper presents a search strategy for moving objects located in a large number of distributed database servers and explores different strategies for updating centralized data. The aim of this work is to process queries such as *where was an object X within a time interval* $[t_i, t_j]$*?* To answer this query, this work proposes to maintain centralized data of the type of objects (e.g., car, truck, train, bus, and so on) that were located within predefined space partitions during time.

The centralized data are organized in a meta-index that contains the *number of objects per type* in each server at different time instants. Thus, the meta-index does not store the actual location of individual objects over space, which could be impractical for applications with high dynamism and very large number of moving objects. The goal of the centralized *meta-index* is to allow the fast determination of the server that contained the object in the near past to the temporal interval of a query. The search starts from this server onwards by setting a software agent to jump from one server to the another until the agent gets all locations of the target moving object during the time interval of interest. To find the starting server, data in the meta-index is used to rank servers based on their likelihood of containing desired data. This ranking of the servers depends on the particular query, because, for a given type of object at a certain instant of time, certain servers can be more important (better ranking) than others as starting points for object searches.

All of this for cases in which it is impractical to maintain information about each object behaviour because of the large number of individuals and the time involved in the collection of information. Nevertheless, the strategies proposed in this paper can be easily adapted to smaller systems in which one can maintain information about every individual stored in a centralized server. Our results show that for these systems it is possible to achieve good performance even for cases in which the central server maintains only partial information about trajectories followed by the individuals across the distributed spatio-temporal database servers. Certainly, a combination of both approaches is feasible, for example, for certain types of inviduals we can resort to maintaining detailed information of trajectories whereas others are treated in bulk in accordance with their specific type classification.

This paper concentrates on defining and evaluating strategies for maintaining up-to-date a centralized meta-index of data about moving objects stored in multiple servers. Two classical approaches for maintaining centralized data are *harvesting*, by which the sources update the copies, or *crawling*, by which the central data repository accesses the sources. These types of strategies can be found in the context of the WWW. Most search engines use a *centralized crawler-indexer architecture* [4], which is associated with a *crawling* strategy for data collection. Particularly interesting for this work are strategies for refreshment [7] [2] and crawling ordering [3] in the WWW, which have been associated with the relevance of web pages. Such relevance have been determined by using change frequency, incoming or outcoming relevant links, and information content.

As such, a crucial assumption about the distributed spatio-temporal database server systems we deal with in this paper, is that they are not supposed to be able to communicate/cooperate between them, namely they are not organized in a specific communication topology and no concept of neighboring servers exists. That is, objects can jump arbitrarely to any server during their lifetimes and can appear/desappear in any of them at any time. In the case of crawling the central server knows addresses of database servers and in the case of harvesting

every server knows the address of the central server. Satisfying this restriction ensures the development of strategies for very large and dynamic systems.

For harvesting the relevant problem to solve is how a particular server can autonomously decide based on local information when to establish a connection with the central server. A sufficiently large number of servers trying to establish a connection with the central server can quickly saturate the available bandwith. A solution can be the use of clock-based synchronous strategies but it is well-known that this type of approaches are not scalable as they can unnecessarely force a no-so-relevant server to communicate with the central one, that is, they fail to put most relevant servers in a given period of time to make use of the bandwith. We propose an asynchronous strategy based on a *communication trigering rule* which also results to be unefficient when applied to harvesting but very efficient when transformed to be used with the crawling strategy.

In a sense we can say that crawling uses *global* information of the recent past to decide what servers should communicate with the central one whereas harvesting uses all *local* information available upto the very present time. As both strategies use the same rule to triger a communication action with the central server and crawling resulted to be by far more efficient than harvesting under demanding benchmark workloads, we claim that recent past information properly combined with a trigering rule and global data for database servers ranking purposes, all kept into the same central server as we propose in this paper can result in a very efficient way of exploiting the communication bandwith.

To the best of our knowledge little research as been done so far on the kind of systems and restrictions we have in mind in this paper. When it comes to keeping track of moving objects, an important number of studies focus on tracking the trajectories of an object, which means that the system knows the actual location of an object at distinct times [18] [17]. Such type of task can also benefit from the results of this work, since finding the starting location of an object within a time interval $[t_1, t_2]$ corresponds to the type of query addressed in this work.

Within the context of handling distributed data of moving objects, Mouza and Rigaux [10] propose a web architecture for scalable moving object servers. They come to the conclusion that it is helpful to divide the space into grids, so that each location of the space is contained in exactly one grid. Their work examines how to answer queries, in which an object enters, crosses or leaves a query window. A similar work that uses partitions of the space was done by Xia and Prabhakar [21]. They propose an index at two different levels: a low level for movements of objects and a high level that organizes the grid of partitions. Both studies based on partitions are focused on window queries and not on queries about locations of particular objects.

The main contributions of this work can be summarized as follows:

- It proposes a search strategy for querying moving objects in a distributed environment.
- It defines two approaches for maintaining the centralized meta-index: *crawling* (i.e., a centralized control of updating) and *harvesting* (i.e., an asynchronous distributed control of updating). This implies the definition of

proper rules for trigering communication and performing the ranking of database servers to be contacted first.
- Unlike previous work related to data replication, this work defines strategies that consider only the local state of servers rather than the state of all other servers. The idea is to minimize the exchange of data between servers and between servers and the centralized system.

The organization of this paper is as follows. Section 2 present the overall system and the search strategy. Strategies for collecting and updating a centralized meta-index are presented in Section 3. Experimental results for comparisons of the harvesting and crawling strategies are shown in Section 4. Conclusions and future research directions are in Section 5.

2 Overall System and Search Strategy

We assume systems composed of several distributed servers that collect information about moving objects. These servers record object movements that occur or have occurred. We aim at a very large number of servers and objects. Each server controls the objects' locations and events (i.e., change of location) within a specific space partition or region. For simplification, the distribution of servers avoids overlapping regions such that the information about an object's location at a specific time instance is stored in only one server.

In this environment, objects move in or move out of regions handled by servers. Unlike previous studies that have addressed queries concerning spatio-temporal windows, such as *what objects were/are in a spatial window at a particular time interval?*, this work aims at providing a method for answering queries of the type *where was a specific object at a given time interval?* This is a particularly challenging query for moving objects with no additional information about previous objects' locations. In a basic system, such query is processed by sending queries to all servers or agents, with overloading consequences for the system. We propose to answer this query with a centralized-access architecture that includes a meta-index that reduces the search space.

For a large-enough distributed database servers system, wherein each server contains large collections of objects of many different types, it is clear that using such a meta-index can reduce the searching time of particular objects by avoiding blind searches that starts, for example, at randomly selected servers.

The basic algorithm we use to conduct searches is as follows,

// Query= where was object o of type c at time t ?.

1. Rank servers according to priority values given by the number of objects of type c at time t in every server. Greater numbers implies higher priorities.
2. Visit every server in ranking order asking for the later event of object o found at time less than or equal to t. Let s be the first server found to have an instance of o.

3. Start the search at server s following the path left by o until reaching the server in which the first move-out event of object o is greater than t. Report solution for Query.

This algorithm can be easily modified to go towards the past or present depending on wether the event nearest to the query time t is a move-out or a move-in respectively.

In table 1 we show results for this strategy and query (we use data from the GSTD generator described in section 4 of this paper). These are results for 10K and 50K moving objects traveling around 100 and 484 nodes (database servers). The column Seq and Prio show results for the average number of visited servers for the blind (Seq) and proposed (Prio) search strategies, respectively The values are the average obtained by performing searches of randomly chosen objects. The column P/S (i.e. Prio/Seq) shows the effectiveness of the proposed searching strategy. [The results in columns Traj and T/P are explained by the end of this section].

Gauss Distribution

Obs	Nds	Seq	Prio	Traj	P/S	T/S
10000	100	39.418	5.104	2.588	0.13	0.07
10000	484	175.476	16.254	4.926	0.09	0.03
50000	100	44.762	4.546	2.452	0.10	0.05
50000	484	199.360	16.110	4.470	0.08	0.02

Uniform Distribution

Obs	Nds	Seq	Prio	Traj	P/S	T/S
10000	100	46.498	26.358	2.494	0.57	0.05
10000	484	216.734	107.834	4.788	0.50	0.02
50000	100	46.354	25.822	2.318	0.56	0.05
50000	484	217.594	107.726	4.412	0.50	0.02

Table 1. Results for searches on a many moving object system.

Certainly the efficiency is crucially dependent on the proper ranking of servers (for simplicity we use the number of objects of a given type but the particularities of real-life systems should expose much more sophisticated ways of defining priorities for servers). This effect can be seen in table 1 looking at the differences P/S between the two spatial distributions (uniform and Gauss) for the initial deployment of moving objects. In the case of the uniform distribution the number of objects per node tends to be similar in every node.

Using the same benchmark system we performed additional experiments for the case in which both the communication bandwith and central server capacity allow one to maintain information about the database servers visited by each

object; we call this information *object trajectories*. In particular, we set our simulator to maintain into the central server upto 20% of the object trajectories provided that for each object at least the initial event is available in the meta-index. As shown in column Traj of table 1, the results are clearly more efficient and stable than in the case of the previous searching algorithm based on type of objects. Column T/S (i.e. Traj/Prio) shows the comparative performance with respect to blind searching (Seq).

Here the searching algorithm is more simple than the one based on object types. Processing a query for object i involves finding the event for i which is closer to the time of the query and then going to the database server s where that event took place and execute the step 3 of the above algorithm based on object types.

Certainly this object-trajectories based searching strategy is much faster than the one based on object-types. However, a clear information transfer cost tradeoff arises when we significantly scale up the overall system. In this case for practical systems we should expect the number of different object types to grow up much more slowly than the number of moving objects.

3 Strategies for Collecting Information

We explore two strategies for collecting data about moving objects, which can be related to the *crawling* and *harvesting* strategies for data replication [13], respectively. Both strategies are based on the idea of representing the need of updating a server's information as a function of time (i..e, a time-varying rank for each server). Three values characterize the state of a server, which are used for determining the server's rank in an ordering policy for updating (priority).

Definition 1 (Normalized variation). *Given the total number of objects in a server i at time instant t_1 ($n(i,t_1)$) and the maximum average number of objects in the server in the time interval $[t_1,t_2]$ ($MaxAvgN(i,t_1,t_2)$), the normalized variation of a server is given by*

$$\alpha(i,t_1,t_2) = \frac{MaxAvgN(i,t_1,t_2)}{n(i,t_1)} \tag{1}$$

Definition 2 (Weighted average of objects). *Given a total number of objects N and the number of objects in a server i at time t ($n(i,t)$), the weighted average of objects in the server for a time interval $[t_1,t_2]$ is given by*

$$\beta(i,t_1,t_2) = \frac{\sum_{\forall t,t_1 \leq t \leq t_2} n(i,t)\Delta(t)}{(t_2-t_1)\times N} \tag{2}$$

Definition 3 (Normalized rate of update requests). *Let be $N_\alpha(i,t_1,t_2)$ the number of times that the normalized variation of a server during a time interval $[t_1,t_2]$ is larger than a tolerance threshold τ (i.e, $\alpha(i,t_1,t) > \tau$, with $t \leq t_2$). The normalized number of rate requests is given by*

$$\delta(i, t_1, t_2) = \frac{N_\alpha(i, t_1, t_2)}{(t_2 - t_1)} \tag{3}$$

Note that the highest values for $\beta(i, t_1, t_2)$ and $\delta(i, t_1, t_2)$ (i.e., 1.0) represent the case when the server concentrates the total set of objects and update requests at all time instances. We consider this case, the highest possible value associated with the importance of updating a server, i.e., the highest rank for data collection (or update).

We represent the state of servers as points in a 2D space, with one of the axis being the weighted average of objects $\beta(i, t_1, t_2)$ and the other axis being the normalized number of update requests $\delta(i, t_1, t_2)$. Using this space, the server rank for data collection is defined by the inverse of the distance between the state point of a server and the maximum reference point (1.0,1.0) (Equation 4, Figure 1).

$$rank(i, t_1, t_2) = 1 - \sqrt{\frac{(1.0 - \beta(i, t_1, t_2))^2 + (1.0 - \delta(i, t_1, t_2))^2}{2}} \tag{4}$$

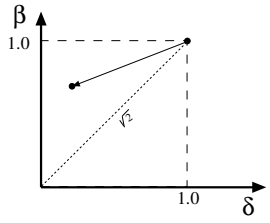


Fig. 1. Rank space

The two strategies for updating the centralized meta-index are:

- **The crawling strategy** follows a similar approach to the *crawling* strategy of WWW search engines [4]. There exists a *scheduler* that makes use of a list of servers to assign the job of retrieving one server to each of a set of so-called *robots*. Normally, the scheduler fetches a server and passes it onto the next idle robot. Once a robot retrieves the data associated with its current server, it also extracts from it statistics data (i.e., number of update requests and current number of objects per type), and passes them to the scheduler. These data are used to re-calculate the ranking of servers in the scheduler's list.
- **The harvesting strategy** makes each database server to control the update of its information in the centralized system. There exists a *harvester* that

receives requests from servers to update information. To trigger the request of an update to the meta-index, each server periodically calculates the α value defined by Equation 1. In the case that the relative variation of values is larger than a tolerance threshold τ, the database server sends an update request to the harvester. If the harvester is idle, it receives the information; otherwise, the server needs to re-try latter.

For the searching strategy based on object-types what is passed on to the central server during a communication action is a representation of the variation function for the number of object of each type observed in the period, whereas for the object-trajectories based one a list of the events associated to every moving object.

Both cases allows the sending of partial information with affecting noticeably the performance of the searching algorithm started at the central server side. For the object-types strategy things are simpler as we only need to reconstruct an approximation of the curve shape.

For the object-trajectories strategy the reduction of the amount of information to be transfered can be more complicated. The strategy we used in the experiments reported in section 2 of this paper is as follows. We reported the central server just one event per co-resident object in the database server and the central server stored for each object only the events separated by more than Δt units of time. The value Δt was set so that the total number of events kept in the central server was on average 20% of the global total number of events in the system.

4 Experimental Evaluations

The problem of gathering data of moving objects in a distributed system is a hard scheduling problem that can be modeled mathematically and simulated as we propose in this paper. Our approach to implementing the crawling and harvesting strategies is based on the use of object-oriented discrete-event simulation models for analyzing the behavior of data collection strategies.

To compare different strategies, we consider the time period when the meta-index is out of date. Consider the Figure 2 describing the changes in the number of objects, per type, in a given server, where at times t_0 and t_n we have data collections for a given node. Between $[t_0, t_n]$, there were 4 events that occurred (2 move_in and 2 move_out events). The time in which the information was out of date before the collection at time t_1 can be calculated as the sum of differences between the time of data collection and time of previous no collected events.

Definition 4 (Divergence metrics). *Given $Ev(i,t)$ the set of events at node i and time instant t, $n(i,t)$ the number of objects in server i at time t, and N the total number of objects, the divergence metrics for the node i in a time interval $[t_1, t_2]$ ($D_n(t_1, t_2)$) is defined as the weighted sum of the temporal gap between events and updates, that is,*

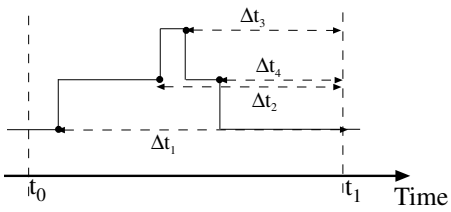


Fig. 2. Period of no updated data

$$D(i, t_1, t_2) = \sum_{\forall t, t_1 \leq t \leq t_2} |Ev(t, n)| * (t_2 - t)) * n(i, t_2)/N \tag{5}$$

In addition to the divergence metrics, we analyze for harvesting strategy the time when the harvester was unable to attend a request for receiving data from a server, i.e., it was busy.

The experiments use two synthetic data about moving objects:

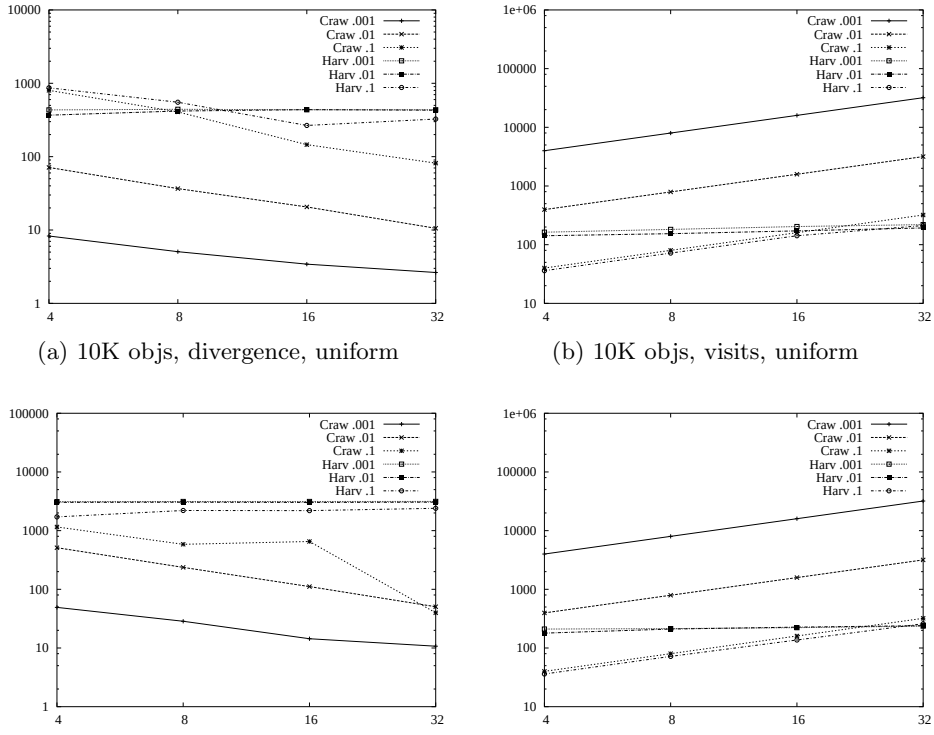
- **Data from the generator GSTD [20].** Sets with 10000 and 50000 points (objects), respectively, were created. These points are distributed uniformly or with a Gaussian distribution in a region at time instant 0.0. Subsequently, they are moved randomly during the next 1000 time instants. Using this data, we created a uniform distributions of servers (100 and 484 servers) that store the movements of objects.

- **A simulated environment of objects and servers.** A number of 100 and 500 servers were distributed uniformly or with a gaussian distribution. Each of these servers was initially set with a random number of objects to obtain 10000 to 50000 moving objects. The regions controlled by servers were defined by using a Delaunay Triangulation [14]. The movements of objects follow the shortest paths between two random servers in the neighboring graph of space partitions.

Results with the GSTD data set. In figures 3 and 4 we show results for the crawling and harvesting strategies. The tolerance *tau*, i.e., the threshold for the observed change of the normalized variation of a server (section 3), was kept at 0.15 in all experiments (similar values were observed for $tau < 0.3$). In the figures *Divergence* is a measure indicating how efficient is a given strategy to update the meta-index. Smaller values indicate better efficiency. We performed experiments for different network latencies, namely 0.001, 0.01 and 0.1. The curve labels are suffixed with 001, 01 and 1 to identify the respective latencies. We also experimented with 4, 8, 16 and 32 bandwiths which means that the central server

can hold 4, 8, 16 and 32 concurrent connexions respectively. These are the values for the x-axis in the figures. The measure *Visits* indicates the number of times a given entity collected data from the system in order to update the meta-index.

The results in figure 3 clearly indicate that crawling is more efficient than harvesting in terms of keeping better updated the meta-index. This stands over the different experiments. This is because the crawling strategy is able to visit more frequently the database servers. Similar results were obtained for 50,000 objects. In figure 4 we show that the harvesting strategy is not able to achieve this because its lack of global synchronization leads to many failed attempts to contact the central server (in those cases the server is busy attending other database servers requests). Only when network latencies are very high the crawling strategy tends to be similar to the harvesting strategy.



(a) 10K objs, divergence, uniform

(b) 10K objs, visits, uniform

(c) 10K objs, divergence, Gaussian

(d) 10K objs, visits, Gaussian

Fig. 3. Uniform and Gaussian distributions for 10,000 objects.

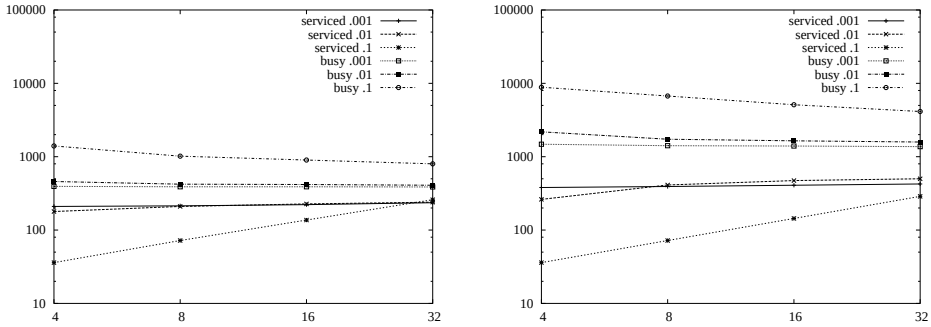


Fig. 4. Gaussian distribution for the harvesting strategy. Curves show the number of attempts to contact the central server in order to update its meta-index :(a) 10,000 objects and (b) 50,000 objects

Results with the simulated servers and moving objects. The results for these experiments are shown in tables 1 and 2. Similar conclusions to the above experiments hold in this case as well.

Table 2. Results with a random uniform distribution

Delay	Connexions	Divergence Crawling	Divergence Harvesting	Visits Crawling	Visits Harvesting	Busy Harvesting
0.1	4	38961	2189827	4000	191	925
0.1	32	3574	2278456	32000	217	395
0.5	4	196699	2083029	797	172	3215
0.5	32	22316	2088935	6369	309	901
1.0	4	393697	1962487	397	148	5994
1.0	32	44385	2118937	3169	359	1591

5 Conclusions

We have presented a strategy to maintain up-to-date a meta-index which allows a centralized server to conduct searches for particular objects in a spatio-temporal moving objects system. We have evaluated two strategies (crawling and harvesting) to collect information from the distributed database servers that maintain track of the moving objects located at a particular area.

We used demanding synthetic work-loads to evaluate the performance of the crawling and harvesting strategies under exactly the same conditions and scenarios. The results show that crawling is more efficient than harvesting as

Table 3. Results with a random clustering distribution

Delay	Connexions	Divergence Crawling	Divergence Harvesting	Visits Crawling	Visits Harvesting	Busy Harvesting
0.1	4	40706	2360123	4000	185	895
0.1	32	4043	2383329	32000	233	415
0.5	4	207966	2198299	797	170	3325
0.5	32	23870	2288753	6369	315	921
1.0	4	416594	2043532	397	162	6126
1.0	32	46907	2282859	3169	350	1571

it keeps the meta-index closer to the real system. This is particularly evident for large number of objects. This advantage stands even in conditions of very high latency in communication of information from the database servers. The reason is that crawling is able to exploit in a more efficient manner the available bandwith for communication.

Acknowledgment. This work has been funded by CONICYT, Chile, under grant FONDECYT 1050944 and UMAG PR-F1-01-IC-04.

References

1. S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *Proceedings of the Rwelfth International Conference on World Wide Web*, pages 280–290. ACM Press, 2003.
2. A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the web. *ACM Transactions on Internet Technology*, 1(1):2–43, August 2001.
3. R. Baeza-Yates, C. Castillo, M. Marín, and A. Rodríguez. Crawling a country: Better strategies than breadth-first for web page ordering. In *WWW05, Industrial Track*. ACM Press, 2005.
4. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley/ACM Press, 1999.
5. S. Chakrabarti. *Mining the Web*. Morgan Kaufmann, 2003.
6. Junghoo Cho, Hector García-Molina, and Lawrence Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.
7. E. G. Coffman, Zhen Liu, and Richard R. Weber. Optimal robot scheduling for web search engines. Technical Report RR-3317, 1997.
8. Edith Cohen and Haim Kaplan. Refreshment policies for web content caches. *Comput. Networks*, 38(6):795–808, 2002.
9. L. Do, O. Ram, and P. Drew. The need for distributed asynchronous transactions. *ACM SIGMOD Record*, 28(2):534–535, 1999.
10. C. du Mouza and P. Rigaux. Web architectures for scalable moving object servers. In *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, pages 17 – 22, New York, NY, 2002. ACM Press.
11. Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, pages 767–778. IEEE Computer Society, 2005.

12. Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages. In *10th World Wide Web Conference*, pages 114–118, Hong Kong, May 2001. Elsevier Science.
13. Christopher Olston and Jennifer Widom. Efficient monitoring and querying of distributed, dynamic data via approximate replication. *IEEE Data Eng. Bull.*, 28(1):11–18, 2005.
14. J. O'Rourke. *Computational Geometry*. Cambridge University Press, 1993.
15. L. Page, S. Brin, R. Motwain, and T. Winograd. The pagerank citation algorithm: bringing order to the web. In *7th World Wide Web Conference*, 1998.
16. G. Pant, S. Bradshaw, and F. Menczer. Search engine-crawler symbiosis. In *Proceeding of the European Conference on Digital Libraries, LNCS 2769*, pages 212–232. Springer-Verlag, 2003.
17. Kostas Patroumpas and Timos K. Sellis. Managing trajectories of moving objects as data streams. In Jörg Sander and Mario A. Nascimento, editors, *STDBM*, pages 41–48, 2004.
18. C. Shahabi, M. Kolahdouzan, S. Thakkar, J. Ambite, and C. Knoblock. Efficiently querying moving objects with pre-defined paths in a distributed environment. In *Proceedings of the 9th ACM international symposium on Advances in geographic information systems*, pages 34 – 40, New York, NY, 2001. ACM Press.
19. Vladislav Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed web crawler. In *ICDE*, 2002.
20. Yannis Theodoridis, Jefferson R. O. Silva, and Mario A. Nascimento. On the generation of spatiotemporal datasets. In Ralf Hartmut Güting, Dimitris Papadias, and Frederick H. Lochovsky, editors, *SSD*, volume 1651 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 1999.
21. Y. Xia and S. Prabhakar. Efficient cng indexing in location aware services. In *23rd International Conference on Distributed Computing Systems Workshops*, page 414. IEEE Press, 2003.