Sync/Async Web Search Engines modelled with TCPN

Veronica Gil-Costa and Marcela Printista

Computing Department, National University of San Luis
Ejrcito de los Andes 950, San Luis , Argentina
Contact e-mail: {gvcosta,mprinti}@unsl.edu.ar

Abstract. The development of distributed systems is particularly challenging because these systems possess concurrency and non-determinism. One way to approach the challenge of developing modern systems is to build a model of the systems. In this paper we model a design of Web Parallel Search Engine (WPSE) using an extension of Petri Net, named Time Colored Petri Nets (TCPN), for a synchronous (Sync) and an asynchronous (Async) communication system using a document partition index . The goal is to simulate the execution of Q queries asking for service at the WPSE for both communication modes. In order to obtain an abstraction of the query processing cost, we take advantage of the TCPN features and modify them in order to perform an performance evaluation independent of the particular hardware platform used.

1 Introduction

Web search systems like Google or Yahoo! provides service to millions of people accessing the Web and searching for information. To speed-up queries answers and to obtain a reasonable response time per query, search engine uses cluster of high performance computers (HPC). Information recovered from the Web is stored as web pages in a repository and the system uses an index to reduce the search cost (in meaning of time and resources use). The index data structure selected in most search engines is the *Inverted File*, which is composed as a set of relevant terms t_i found in the pages with a posting list $[t_i, < d_j, f_{t_i,d_j} >]$. The posting list of each term has the information required to identify the document d_j where the term appears and the frequency f_{t_i,d_j} of the term in that document. The frequency is used during the search phase to obtain the relevance of the document for a query.

There are two main schemes to perform the parallel query searches: document partition and term partition. In the first one all documents are distributed among a set of P processors, and then each one builds its own inverted file index using the local documents. Therefore, if the collection has N documents and T terms, each processor will hold N/P documents and the inverted file will have T terms. In the term partitioned scheme, we build one sequential index and then we distribute each term with its whole posting list among processors. So in this case each processor will have N documents but only T/P terms. There are

many variants of these strategies like [14, 16, 24], but the main discussion remains between the term and the document partitioned schemes. Some papers presents good results for the document partitioned scheme while others argue that the term partitions is more scalable. What is out of discussion is that the document partitioned has the advantage of a very cheep construction and upload, because no communication is required during these process.

Most implementations of distributed inverted indexes reported so far are based on the message passing approach to parallel computing in which we can even see combinations of multi-threaded and computation/communication overlapped systems. We believe that under such heterogeneous forms of parallel computing it is quite risky to make reasonable claims about comparative performance as it is difficult to reproduce the same experimental conditions. For example, the execution of the system is dependent on the particular state of the machine and its fluctuations. On the other hand, artefacts such as threads are potential sources of overheads that can produce unpredictable outcomes in terms of running time. Yet another source of unpredictable behaviour can be the accesses to disk used to retrieve the posting lists. In fact, the difficulty of comparing these kind of systems is discussed in [15].

Search operations are performed upon a cluster of machines in order to reduce the algorithm execution time. In the literature most papers presents asynchronous algorithms like [1, 10, 14, 16, 19, 23, 25] where queries are solved using a set of machines but there is no synchronization among them and they do not share the global information system state. In this case, the parallel algorithms are implemented using the pvm [4] or the mpi [20]library. Some others researches like [6, 12, 13] use a synchronous parallel model where it is possible to manage fairly the resource use. In this case all processors must synchronize at a certain time before they continue with their operations. A synchronous system performs well under low query traffic, but when the traffic is high enough these systems tends to loose performance, because it has to pay the latency network cost for each message. Actually Web searches overcome this problem using hardware redundancy, but this solution has no future in terms of algorithms, due to the Web size duplicates every eight months. On the other hand, synchronous systems tends to perform better with a high query traffic, because only one message is sent per processor and the latency network cost is lower. Some experiments proving this claims are presented in [12, 11, 2, 5].

In this paper we present a design of Web Parallel Search Engine (WPSE) using a Petri Net [18] for a synchronous (Sync) and an asynchronous (Async) system using a document-partition scheme and we compare them under the same rules. So we can simulate the execution of Q queries asking for service at the search engine for both modes of communication. The novelty of this paper is based in the use of an extension of Petri Nets named Time Colored Petri Nets (TCPN) [7] to model a parallel problem discussed for many years and we present a modification to this network model tool in order to obtain our goal.

This paper is organized as follows. Section 2 presents and describes TCPNs. Section 3 presents the query processing algorithm. Section 4 applies TCPNs to

model and to obtain the cost of a synchronous web search engine and Section 5 for the asynchronous communication mode. Finally, Section 6 presents the conclusions and futures works.

2 Petri Nets

Petri Nets, developed in the early 1960s by Carl A. Petri to model concurrent computer system operations, most widely accepted and well-known graphic approaches in modeling discrete event systems (DESs) including various discrete manufacturing systems. Actually, PNs are also supported by sophisticated mathematics.

In the use of PNs for modelling real systems, several authors have found convenient to introduce special constructs either for making the model representation more compact in a given application or for extending the modelling power of the PN formalism. Among extensions to PNs, timed coloured Petri Nets (TCPNs) [7, 8] is one of high level PNs which has advantages over basic PNs in making the model of a DES much more compact and concise. This is gained by introducing **colours** to distinguish among tokens presenting different entities (e.g. parts to be processed) and using additional logic (arc expression functions) to control token flows [9].

In TCPNs, a global clock is introduced for timing, and the value of the clock is called model time. A time stamp is defined as the earliest model time at which the token carrying it can be used for enabling a transition TCPNs are based on semantics made by Jensen [1,2]. However, some of these semantics are simplified when TCPNs are used for modelling manufacturing systems as shown in [4]. The TCPN model of the system can be formally described by TCPN semantics as follows:

$$CPN = (\Sigma, P, T, A, C, F, I, O, M_0) \tag{1}$$

where $P = \{p_1, \ldots, p_k\}$ is the set of places (non empty and finite); $T = \{t_1, \ldots, t_k)$ is the set of transitions (non empty and finite); A is the finite set of arcs. $\Sigma = \{A_0, \ldots, A_m, e\}$ is the finite set of non-empty types, also called colour sets, where A_m, denotes the states of part A, while e represents that a resource is idle and available. $C = P \rightarrow \Sigma$ is a colour function. They specify attributes of tokens in a specific place. I and O are a sets of input and output arcs. M_0 are a set of initial markings. F is a guard function. It is defined from T into boolean functions, that is, their evaluations are either TRUE (gate is open) or FALSE (gate is closed) [3]. They prevent tokens of specific colours from flowing through a transition. Given a marking M, a PN transition is enabled if, beside the normal enabling requirements, the conditioning function is true. The conditioning functions can be very effective in reducing the graphical complexity of a PN, even if they do not extend the modelling power with respect to inhibitor arcs or priority levels. In a TCPN, time can be associated to places or to transitions. In this work we apply the second option.

In this paper we take advantage of the TCPN features and we modify them so the time associated to each transition is deterministic and we represent it as

an abstraction of the query processing cost. The cost assigned to each transition depends exclusively in the number of tokens involves. Therefore we adapt the TCPN representation in order to perform an evaluation independent of the particular hardware platform used.

3 Query Processing

We adapt the Time Coloured Petri Nets [22], to analyze the asymptotic cost of processing Q queries using a document partitioned index in a Web search engine. Queries arrive to the processors from a receptionist machine that is called *broker*. Different realizations of this general scheme have been proposed in the literature. Some are unsuitable for comparison experiments as they hide costs such as communication between the broker and the processors or keep at the broker side part of the ranking cost. Also, in some cases is unclear as to whether the broadcast required by the document partitioning approach is implemented using efficient algorithms. Others interleave posting list fetching and ranking across processors but their cost is equivalent to the strategy we use in this paper.

An innovation presented in [11] is to de-couple the actions of fetching lists from the index and ranking sets of documents by distributing the workload in a round-robin manner. Each processor has a dual role at all times: *fetcher* and *ranker*. In the first role the processor is fetching posting lists from its local index for some query. In the second role, the processor is merging lists to compute the ranking associated to another query. The broker machine is in charge of assigning a processor to be the ranker for each query in a circular manner. The ranker then broadcasts the query to P fetchers in the cluster, where P is the number of processors. Each ranker returns later the corresponding answers to the broker. Every query is then processed using two major steps: the first one consists on each fetcher sending a K/P-sized piece of every posting list involved in the received query to its ranker processor. The size K/P will be chosen such that the ranker gets about K documents in total. In the second step, the ranker performs the actual ranking of documents and, if necessary, it asks for additional K/P-sized pieces to the fetchers in order to produce the K best ranked documents that are passed to the broker as the query results. In fact, the number of answers shown to the user can be less than K and we have find that in practice we need K to be less than twice the number of final answers.

We assume a situation in which the query arrival rate in the broker is large enough to let the broker distribute $Q\,P$ queries onto the P processors. The specific way we organize fetching and ranking along with its bulk-synchronous realization allows to compare in a fair manner the document partitioned approach with others strategies

We use the vector method for performing the ranking along with the filtering technique proposed in [17]. Consequently, the posting lists are sorted by frequency in descending order and we study cases in which the posting lists are intersected before the ranking to determine the documents containing all of the

query terms, as is typical in current search engines, as well as cases in which this operation is not required.

```
while(true)
{
 //Receive Messages
 msg ← queue_in.front();
 switch (msg → type)
 {
    case MSG_BROKER:
        broadcast(msg);
        break;
    case MSG_FETCHING:
          Fetch_Inverted_List(msg,new_msg);
          Send(new_msg);
        break;
    case MSG_RANKING:
        if (wait_Results(msg))
        {
          Ranking(msg,new_msg);
          Send(new_msg);
        }
        break;
 }
}
```

Fig. 1. Queries processing algorithm.

4 Synchronous Query Processing

For synchronous query processing we use the bulk synchronous model of parallel computing (BSP) and its cost model [21]. In BSP the computation is organized as a sequence of *supersteps*. During a superstep, the processors may perform computations on local data and/or send messages to other processors. The messages are available for processing at their destinations by the next superstep, and each superstep is ended with the barrier synchronization of the processors. The underlying communication library ensures that all messages are available at their destinations before starting the next superstep.

A key issue in stable performance is to ensure that each query is given a fair share of the hardware resources in a round-robin manner. To support the round-robin principle we divide query processing in "atoms" of size K, where K is the number of documents delivered to the users. These atoms are scheduled in a round-robin manner across supersteps and processors. Namely, queries are given K sized quanta of processor time, communication network and disk accesses.

These quanta are granted sequentially to queries during supersteps. New queries are injected as soon as the same number of current queries have finished. In the Async mode, the round-robin principle is emulated by performing proper thread scheduling at each processor to grant each active query its respective quantum of execution in a circular manner.

Figure 2 shows a TCPN for a cluster with three processors. We assume a situation of high query traffic and each processor performs a sequence of operations depending on the type of the token. There are three kinds of tokens: x represents a broadcast token, y represents a fetching token, and z represents a ranking token. Function $f()$ indicates the type and number of token received by each state or transition. With $f(x)$ we represent "the type of a token must be x". To simplify the figure, we only show the value of $f()$ for the input of transition. Transitions t_1, t_2 and t_3 represent the operation of extracting messages from the input queue in each processor, and place the token in the correct state. Transition t_4 involves the message generation operation transforming tokens of type x into y tokens (these messages will be broadcasted). Transition t_5 represents the fetching operation transforming tokens of type y into z tokens. Transition t_6 represents the ranking operation, and if the query requires another iteration, the transition transforms the z token into a y token. Otherwise, the transition transforms the token into an m token that will be consumed by transition t_8. transition t_7 represents the synchronization barrier among processors. At this point, all messages are packaged and each processor sends only one message to each other processor (including itself) through the network. Notice that t_7 is enabled only when there are some tokens in p_5, p_6 and p_7, but all tokens in p_2, p_3 and p_4 are consumed. This last condition is establish by the inhibitor arcs. Therefore, using a synchronous communication mode, we have to complete all broadcast, fetching and rankings operations before to continue with the next superstep.

As we said before, in this work me modify the concept of a TCPN associating a cost to each transition depending on the number of queries been processed instead of a real time. The number of queries is represented by the tokens available in the TCPN network. Figure 3 shows the processing of Q queries in one processor across several supersteps using the TCPN (the same operations are processed by all processors). We assume that $q = 2$ queries are injected per superstep so we add a new transition t_0 to represent this action. In the first superstep, transition t_7 only requires tokens from place p_5. These restriction is given by the function $f(y) > 0, f(x) >= 0$ and $f(z) >= 0$. From the third superstep on, we assume a situation with high work load and t_7 requires tokens of type y placed in p_5 and p_7, and at least one token of type z placed in p_6.

The elements composing the TCPN for a Sync web search engine are the following:

$$\begin{aligned}\textstyle\sum &= \{x, y, z, m\} \\ P &= \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\} \\ T &= \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\} \\ M_1 &= \{q, 0, 0, 0, 0, 0, 0\}\end{aligned}$$

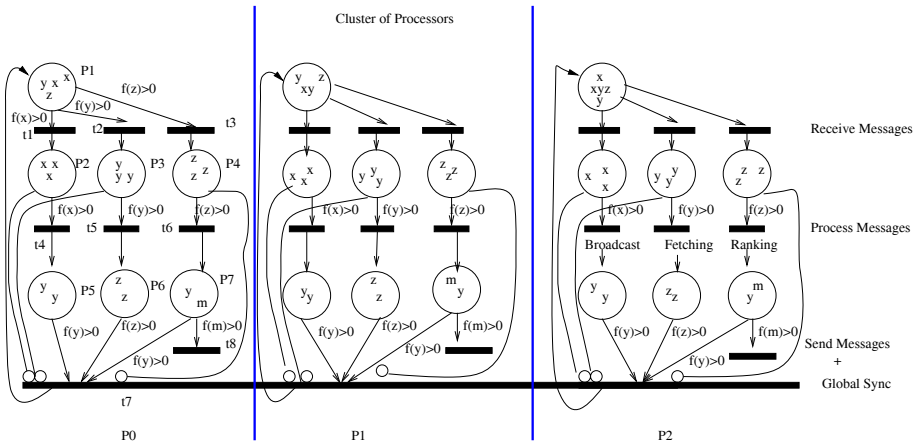

Fig. 2. Cluster view of the queries processing operation with three processors and a global barrier synchronization.

$$
\begin{array}{llll}
C(p_1) = \{x, y, z\} & C(p_2) & = \{x\} & C(p_3) = \{y\} \\
C(p_4) = \{z\} & C(p_5) & = \{y\} & C(p_6) = \{z\} \\
C(p_7) = \{y, m\} & & & \\
C(t_1) = \{x\} & C(t_2) = = \{y\} & & C(t_3) = \{z\} \\
C(t_4) = \{x\} & C(t_5) & = \{y\} & C(t_6) = \{z\} \\
C(t_7) = \{z, y\} & C(t_8) & = \{m\} & \\
I(t_1) = \{p_1\} & I(t_2) & = \{p_1\} & I(t_3) = \{p_1\} \\
I(t_4) = \{p_2\} & I(t_5) & = \{p_3\} & I(t_6) = \{p_4\} \\
I(t_7) = \{p_5, p_6, p_7\} & I(t_8) & = \{p_7\} & \\
O(t_1) = \{p_2\} & O(t_2) & = \{p_3\} & O(t_3) = \{p_4\} \\
O(t_4) = \{p_5\} & O(t_5) & = \{p_6\} & O(t_6) = \{p_7\} \\
O(t_7) = \{p_1\} & O(t_8) & = \{\} &
\end{array}
$$

Figure 4 shows the analytical cost of processing q new queries per superstep using the TCPN presented in Figure 3. The analysis is shown for one processor until N supersteps are executed and the work load in the system is big enough. Due to all processors perform the same sequence of operations we present the analysis for only one processor. N is the number of iterations or supersteps in the BSP model, P is the number of processors in the server, q is the number of queries injected per superstep (that is why always the cost of $t_0 = q$), $\frac{K}{P}$ is the inverted file size (number of documents with the associated frequency) retrieved per query, and $logP + l$ is the synchronization cost plus the latency cost of the network. We assume that every time a ranking operation is performed half of the queries $q/2$ requires a new iteration and the other half finish. Notice that only in t_7 messages are send though the network and we have to pay a latency at most $P - fold$.

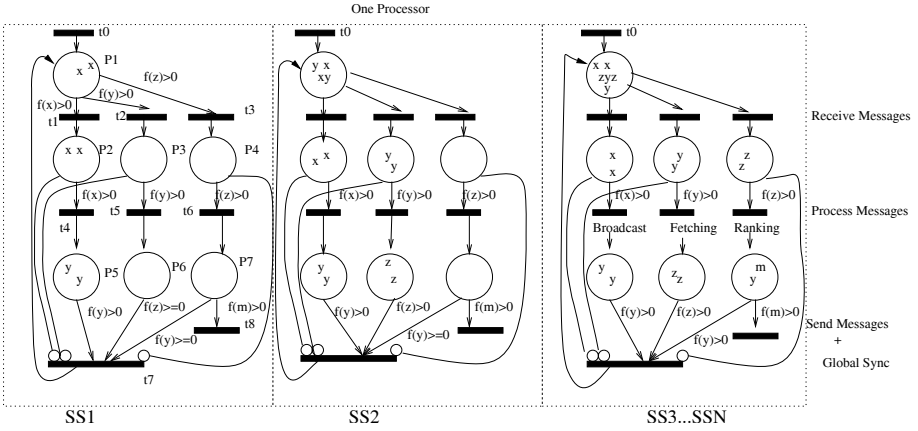

Fig. 3. Superstep view of the query processing operation in one processor.

The cost of t_7 is given by the message size plus the synchronization cost ($logP$) plus the network latency cost which is Pl because we send only one message per processor per superstep. In each superstep SS_i, $\mathbf{F}$ represents the fetching operation and $\mathbf{R}$ represents the ranking operation. Finally, it is important to explain that transition t_3 always receives the fetching results from P processors as we are modeling the document partition inverted file, and therefor transition t_6 is affected directly by t_3 cost. Therefore, the final cost of processing a batch of queries in the Sync Model is given by:

$$C_{SM} = q + q\frac{N}{2}(1 + K + \frac{K}{P} + P) + \frac{N}{2}qK\mathbf{F} + \frac{N}{2}K\mathbf{R} + qP + logP + Pl \quad (2)$$

This analysis is obtained using an initial marking $M_1 = \{q, 0, 0, 0, 0, 0, 0\}$ with q elements of type x placed in p_1, and executing the Petri Net designed through a Sync communication model. For that, in the first superstep, the cost of $t_0 = q$ (always send q new queries). The cost of extracting q messages from the input queue is $t_1 = q$. In this first superstep, $t_2 = t_3 = 0$ because there are no fetching or ranking kind of message and therefore $t_5 = t_6 = t_8 = 0$. The cost of generating P message for each query is $t_4 = qP$. Finally the t_7 is given by the size of message being send through the network (qP), plus the synchronization cost ($\log P$), plus the latency cost of sending P messages. This analysis is repeated for the next supersteps.

Also, we can obtain the reachability graph $G_R(SM)$ as shown in Figure 5. Such reachability graph contains a node for each possible state and an arc for each possible state change.

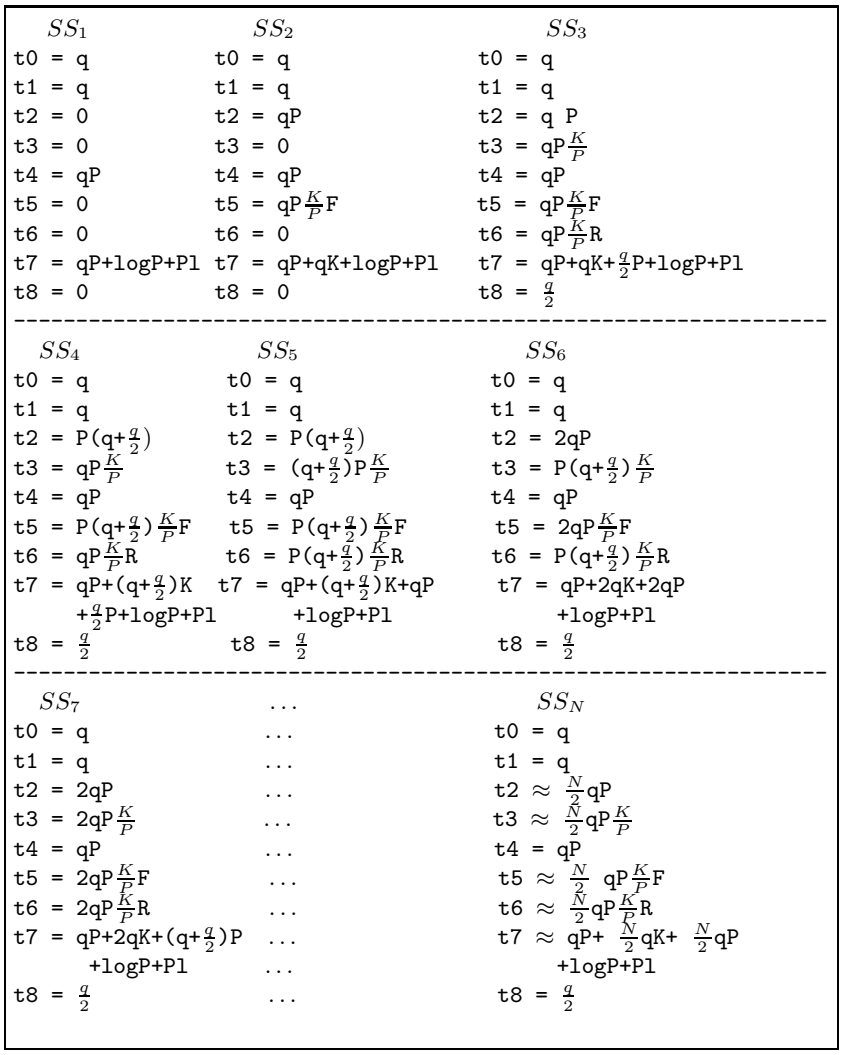

Fig. 4. Analytical cost for executing Q queries with a Sync Web searcher.

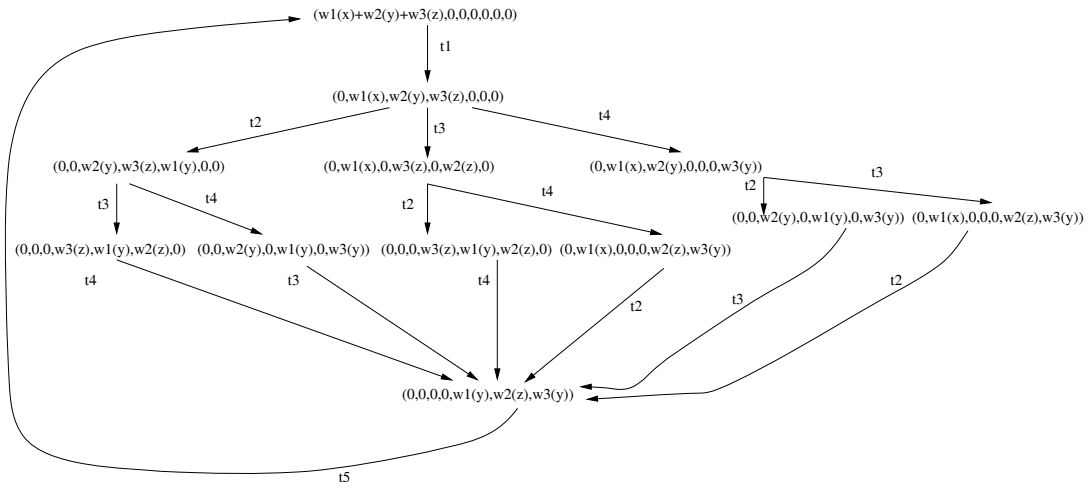

Fig. 5. Reachability graph for the TCPN represented in Figure 3. Symbols $\omega 1$, $\omega 2$ and $\omega 3$ an arbitrary number of tokens x, y y z.

5 Asynchronous Query Processing

In the Asynchronous communication mode, queries are solve using the same scheme presented in Section 3 but without a periodic synchronization after each iteration. Therefore, processors process queries, send and receive messages regardless others processors and without blocking themselves.

Figure 6 shows the design obtained for an Async Web search engine using a TCPN. Operations performed by the Async transition are the same as described for the Sync communication mode. And the elements of the Async TCPN are:

$$\begin{aligned}
\textstyle\sum &= \{x, y, z, m\} \\
P &= \{p_1, p_2, p_3, p_4, p_5\} \\
T &= \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\} \\
M_1 &= \{q, 0, 0, 0, 0\}
\end{aligned}$$

$$\begin{array}{lll}
C(p_1) = \{x, y, z\} & C(p_2) = \{x\} & C(p_3) = \{y\} \\
C(p_4) = \{z\} & C(p_5) = \{y, z, m\} & \\
C(t_1) = \{x\} & C(t_2) = \{y\} & C(t_3) = \{z\} \\
C(t_4) = \{x\} & C(t_5) = \{y\} & C(t_6) = \{z\} \\
C(t_7) = \{y, z\} & C(t_8) = \{m\} & C(t_0) = \{\} \\
I(t_1) = \{p_1\} & I(t_2) = \{p_1\} & I(t_3) = \{p_1\} \\
I(t_4) = \{p_2\} & I(t_5) = \{p_3\} & I(t_6) = \{p_4\} \\
I(t_7) = \{p_5\} & I(t_8) = \{p_5\} & I(t_0) = \{\} \\
O(t_1) = \{p_2\} & O(t_2) = \{p_3\} & O(t_3) = \{p_4\} \\
O(t_4) = \{p_5\} & O(t_5) = \{p_5\} & O(t_6) = \{p_5\} \\
O(t_7) = \{p_1\} & O(t_8) = \{\} & O(t_0) = \{p_1\}
\end{array}$$

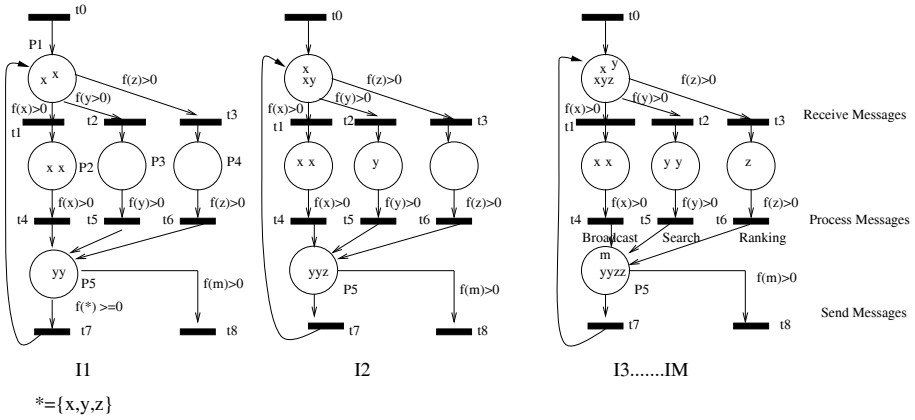

Fig. 6. Asynchronous query processing using a TCPN.

We perform a cost analysis using the TCPN and with an initial marking $M_1 = \{q, 0, 0, 0, 0\}$ with q elements of type x placed in p_1 as shown in Figure 7. In this analysis we assume that all processors have the same process speed and they have to deal with a high query traffic (q new queries are injected per iteration). The reachability graph $G_R(AM)$ is obtained in a similar way as we shown in Figure 5.

In the Async mode each message is sent without waiting for a barrier, so each time a message is sent though the network we have to pay a latency cost l. We have found that for high query traffic the Async communication mode has to pay a high latency cost [12], and it can significantly affect the performance of the system. So assuming that the ranking operation is the most expensive one, we can obtain the cost of processing a batch of q queries as shown in Figure 7.

And the asymptotic cost is given in equation 3. In this equation we can see how the network latency affect the efficiency of the Async communication model, and comparing it with the Sync mode, the cost is highly dependent on the number of active queries in the system waiting for being completed.

$$C_{AM} = q + q\frac{N}{2}(1 + K + \frac{K}{P} + P) + \frac{N}{2}qK\mathbf{F} + \frac{N}{2}K\mathbf{R} + qP + (qP + \frac{N}{2}q + q\frac{N}{2}P)l \quad (3)$$

6 Conclusions

To cope with the complexity of current web system, it is crucial to provide methods that enable the studying of central parts of the system designs prior to implementation. In this paper we have used TCPN to model a Web Parallel Search Engine and analyse two communication system. The TCPN model uses a new interpretation of T (time) where we represent the time associated to each

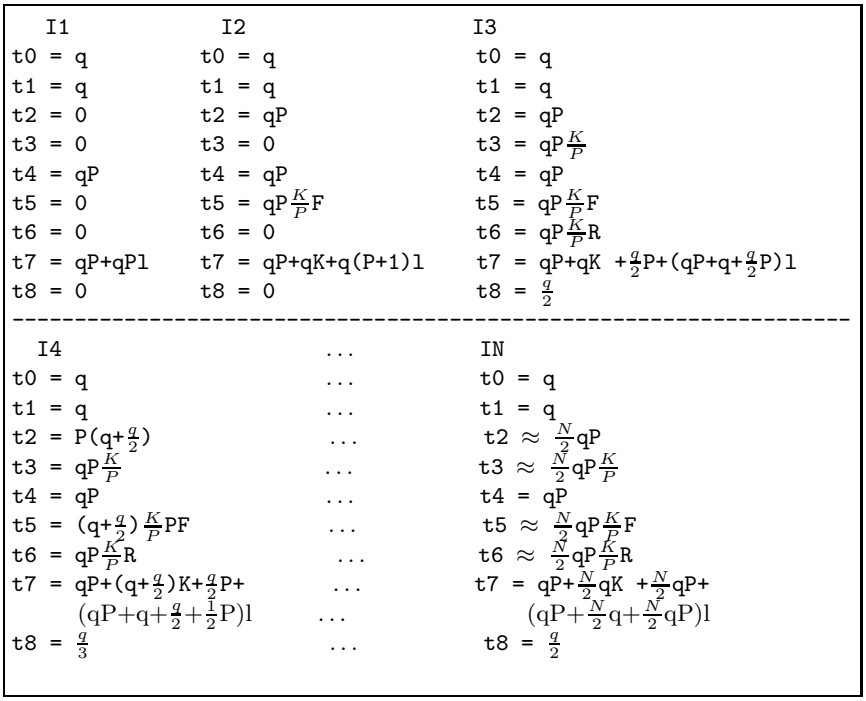

Fig. 7. Analytical cost for executing Q queries with a Async Web searcher.

transition as an abstraction of the query processing cost. The model based on TCPN proposed in this work evidences the involved cost for each communication mode. In the Sync communication mode it is clear that when the query traffic is low, the synchronization cost is higher and therefore the performance will be lower. On the other hand, when the Async communication mode is used and the query traffic is high it has to pay a higher cost due to the network latencies.

Acknowledgments

We wish to thank the Universidad Nacional de San Luis, the ANPCYT and the CONICET from which we receive continuous support.

References

1. C. Badue, R. Baeza-Yates, B. Ribeiro, and N. Ziviani. Distributed query processing using partitioned inverted files. *Eighth Symposium on String Processing and Information Retrieval (SPIRE'01)*, pages 10–20, Nov. 2001.
2. G. Costa, M. Marin, and N. Reyes. An empirical evaluation of a distributed clustering-based index for metric space databases. *In International Workshop on Similarity Search and Applications (SISAP 2008), IEEE-CS Press*, Cancun, Mexico, April 11-12, 2008.
3. J. B. Dugan, A. Bobbio, G. Ciardo, and K. Trivedi. The design of a unified package for the solution of stochastic petri net models. *In Proceedings International Workshop on Timed Petri Nets, Torino (Italy) IEEE Comp Soc*, pages 6–13, 1985.
4. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Network Parallel Computing*, 1994. MIT Press.
5. V. Gil-Costa, M. Marin, and N. Reyes. Parallel query processing on distributed clustering indexes. *Journal of Discrete Algorithms (Elsevier)*, 7:3–17, 2009.
6. C. Gomez and M. Marin. Load balancing query ranking on distributed inverted files. *In 16th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2008), IEEE-CS Press*, pages 329–333, Toulouse, France, Feb. 13-15, 2008.
7. K. Jensen. Colored petri nets: Basic concepts, analysis methods and practical use. *Springer Volume 1*, New York, 1992.
8. K. Jensen. Colored petri nets: Basic concepts, analysis methods and practical use. *Springer Volume 2*, New York, 1995.
9. Z. B. Jiang. *Petri Nets with changeable structure for modeling and adaptive control of one-of-a-kind production systems*. Ph.D. thesis, City University of Hong Kong, 1999.
10. C. Lucchese, S. Orlando, R. Perego, and F. Silvestri. Mining query logs to optimize index partitioning in parallel web search engines. In *In Proceedings of Infoscale Suzhou, China, June 06 - 08*, 2007.
11. M. Marin, C. Bonacic, G. Costa, and C. Gomez. A search engine accepting online updates. *In 13th European International Conference on Parallel Processing (Euro-Par 2007)*, 4641:340–349, Rennes, France, Aug. 28-31, 2007.
12. M. Marin and V. Gil-Costa. (sync-async)+ mpi search engines. *EuroPVMMPI (Lecture Notes in Computer Science, Springer-Verlag)*, 2007, Paris, France, Oct. 2007.

13. M. Marin, C. Gomez, S. Gonzalez, and G. Costa. Scheduling intersection queries in term partitioned inverted files. *In 14th European International Conference on Parallel Processing (Euro-Par 2008)*, 5168:434–443, Gran Canaria, Aug. 26-29, Spain, 2008.
14. A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 348–355, 2006.
15. A. Moffat and J. Zobel. What does it mean to measure performance? *Proc. 5th Int. Conf. on Web Informations Systems,LNCS 3306,Springer*, pages 1–12, Brisbane, Australia,2004.
16. W. Moffat, J. Webber, Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 348–355, September 2005.
17. M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47(10):749–764, 1996.
18. C. Petri. Kommunikation mit automaten. In *Technical report, Doctoral Thesis, University of Bonn. Available in English as: Communication with automata, Technical Report RADC-TR-65-377, Rome Air Development Center, Griffiss NY, 1966*, 1962.
19. B. Ribeiro-Neto, J. Kitajima, G. Navarro, C. Santana, and N. Ziviani. Parallel generation of inverted lists for distributed text collections. In *XVIII Conference of the Chilean Computer Science Society*, pages 149–157. (IEEE CS Press), 1998.
20. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The complete Reference*, 1996.
21. L. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33:103–111, Aug. 1990.
22. W. M. P. van der Aalst. Interval timed coloured petri nets and their analysis. In *Application and Theory of Petri Nets*, pages 453–472, 1993.
23. I. H. Witten, A. Moffat, and T. Bell. *Managing gigabytes: Compressing and indexing documents and images*. 2nd ed. San Francisco, Morgan Kaufmann, 1999.
24. W. Xi, O. Sornil, M. Luo, and E. A. Fox. Hybrid partition inverted files: Experimental validation. In *ECDL '02: Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, pages 422–431, London, UK 2002.
25. J. Zhang and T. Suel. Optimized inverted list assignment in distributed search engine architectures. *In IPDPS*, 2007.