

A CPN Model of an Internet Object Cache

B. Kolics and K. M. Hangos

Computer and Automation Research Institute Hung. Acad. Sci.

H-1518 Budapest P.O. Box 62, Hungary

E-mail: Bertold.Kolics@sztaki.hu, hangos@scl.sztaki.hu

Abstract

A request-level timed stochastic CPN model of a leaf Internet object cache is proposed in this paper. The leaf cache is modelled as a discrete time discrete event dynamic system together with its adjustable input-, measurable output, disturbance and state variables.

The CPN model has been verified and validated against real measured data. The developed model will be used form model-based optimal selection of the most influential tuning parameters modelled as input variables on the performance of the cache.

1 Introduction

Internet object caching is a widely used technique in the current Internet environment. The primary function of object caches is to store popular objects (originating from Web, FTP, Gopher servers in general) and in this way they reduce repetitious accesses to the same resource. As a result, both network bandwidth is saved and users perceived latency is reduced.

Several object caching software products exist which provide a rich set of configuration settings, so cache administrators can easily configure the software products. However, the initial configuration settings may not reflect the given state of network, operating system and other cache influencing parameters, i.e. these settings represent rather a static than a dynamic model. Some articles has already claimed that adaptive approach outperforms other, non-dynamic cache models [1],[2].

Therefore the aim of the project was to develop a *request-level CPN model* of a leaf cache, to develop methods for its verification and validation against

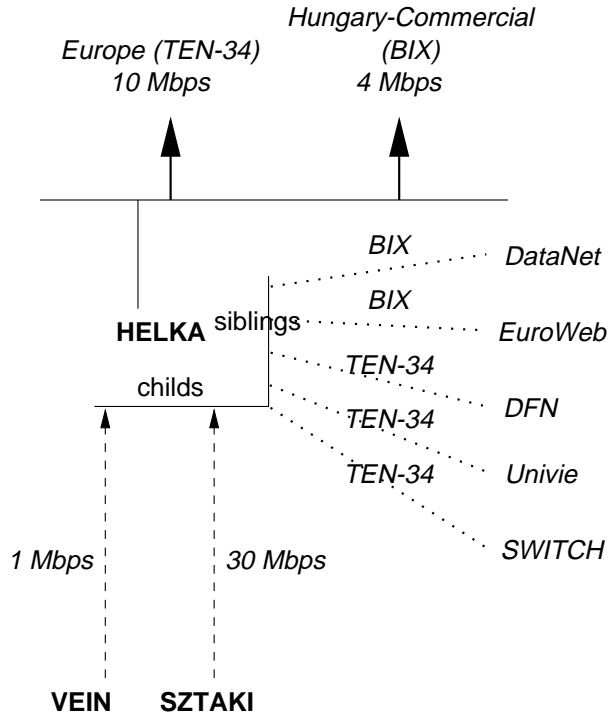


Figure 1: Cache hierarchy used for modelling a leaf cache

measured data in order to be able to investigate the effect of the most influential tuning parameters on the performance of the cache.

The paper is organized as follows. First the modelled system, the leaf cache is described as a dynamic system together with its input, output, state and disturbance variables. Thereafter the CPN model of the cache is developed starting from the modelling assumptions with all of its elements: hierarchy pages, places and transitions. Thereafter the methods and results of the model verification follows and some simulation results are shown. Finally conclusions are drawn.

2 The Internet Object Cache

2.1 System description

A simplified dynamic model of a leaf cache in a cache hierarchy is developed. The cache server has a parent on a higher level. The cache hierarchy with the place of the investigated leaf cache server is shown on Fig. 2.1.

The cache accepts requests from the clients and delivers the requested

objects to them either from its cached objects or downloaded either from the hierarchy or from the original source.

If the requested object is not present in the leaf cache then it sends a query to the hierarchy, in this case to its parent, to deliver it. It is assumed that after a given time-out the cache is able to download the requested object itself from the original source.

2.2 The modelling goal

The aim of our modelling is to find ways to tune the most influencing cache tuning parameters in a model-based and adaptive way. Therefore the model includes:

- the cache tuning parameters which influence the performance of web caches primarily,
- the relevant set of state variables which describes the whole caching process,
- the change of relevant internal cache variables over time,
- the most important measurable variables characterising the operation and performance of the cache.

Because of the above modelling goal a *request level mechanistic dynamic model* is needed which is able to describe the non-linear dynamic behaviour of the cache over a wide operation region.

2.3 Dynamic (time varying) variables

The performance of a proxy cache is characterised by the *hit-rate* and by the *response time* of the individual requests. Moreover, the quality of service based on cached WWW-objects is also important where the ratio of stale (i.e. *not up-to-date responses*) is of importance. The above variables can be regarded as *measurable output variables* of the proxy cache as a dynamic system and will naturally be the subject of optimisation or control.

The major dynamic variables influencing the behaviour of a proxy cache are that of the characteristics of the requests. The *number of requests per hour* is a time-varying and random variable which can be regarded as a *disturbance* to the dynamic system. Its average value characterises the *load of the cache*. However, the quality of the requests (the “difficulty” of the requests) also plays role but it is rather difficult to associate a measurable variable (or variables) to it.

The set of *state variables* of the proxy cache system consists of the variables which characterise the actual status of the cache comprising its “past” into their value. The actual status of the resources of the proxy cache computer relevant to caching play role here, such as *the size and the content of the cache memory and disc space, the content and organisation of the object disc directories* etc. The state variables in our CPN model reflect the current state of the caching system using a reduced set of cache-related variables and parameters.

Finally, the potential *input variables*, i.e. the measurable variables which can be directly influenced, are surely among the 100 tuning knobs available for tuning proxy caches. From this set the most influencing and important variables are *the available memory size, disk size, the processor capacity, the sizes of the object disc directories, file handlers and the disc speed*.

3 The CPN model

3.1 Modelling assumptions

In order to obtain a simple and feasible model of the caching system the following *assumptions* have been made.

1. The cached objects are placed in a set (without ordering).
2. The query of parent and neighbours is performed in a broadcast manner with specified time-out.
3. The name resolution procedure is not described, it is assumed that its impact is negligible.
4. If there is no response from the hierarchy within the specified time-out then the object is retrieved directly from the source server.
5. The processing of the required freshness of the requests (as specified by the client) is not described.
6. The transmission time of the request from the client is assumed to be negligible compared to the service time of the other processing steps therefore this processing step is not modelled.
7. The effect of the hierarchy related tuning parameters is not investigated therefore random variables are used to describe the average behaviour of the hierarchy towards the cache.

With the assumptions above a hierarchical stochastic timed CPN model [6] is developed which is described in a top-down manner.

The model has been implemented and tested using the Design/CPN package [7] containing a CPN editor, a macro-language for declarations and procedures and a CPN simulator. The model is described using the syntax of the Design/CPN package. The model pages of the CPN model are described in details below. The contents of the declaration node is found **in the Appendix..** The PAGE IDENTIFIERS of the CPN model are denoted by SMALLCAPS, **places** by **bold** and *transitions* by *italic* letters.

3.2 Most abstract view of the cache model: the SUPERPAGE

This page contains the top hierarchy page of the model with the following main elements: places and transitions.

Arrival	arrival of requests (input from a file)
<i>Registration</i>	a unique identifier is given to the requests
Registered	registered requests, each request has a unique (URL, identifier) tuple
<i>Is it in the cache?</i>	check if the given URL is present in the cache
Existence tested	requests are marked for local or remote processing
<i>Local object processing</i> (substitution transition)	for requests present in the cache detailed description of the processing steps is on page LOCAL
Local object processing done	request has been processed locally (with appropriate time delay)
<i>Transmit to client</i>	with random time delay
<i>Remote object processing</i> (substitution transition)	for requests missing from the cache (the detailed description of the processing steps is on page REMOTE)
Exit	end of request processing (output to a file).

3.3 Initialization: page INPUT

The timed requests are read from a file on this page and are transmitted to the place ARRIVAL.

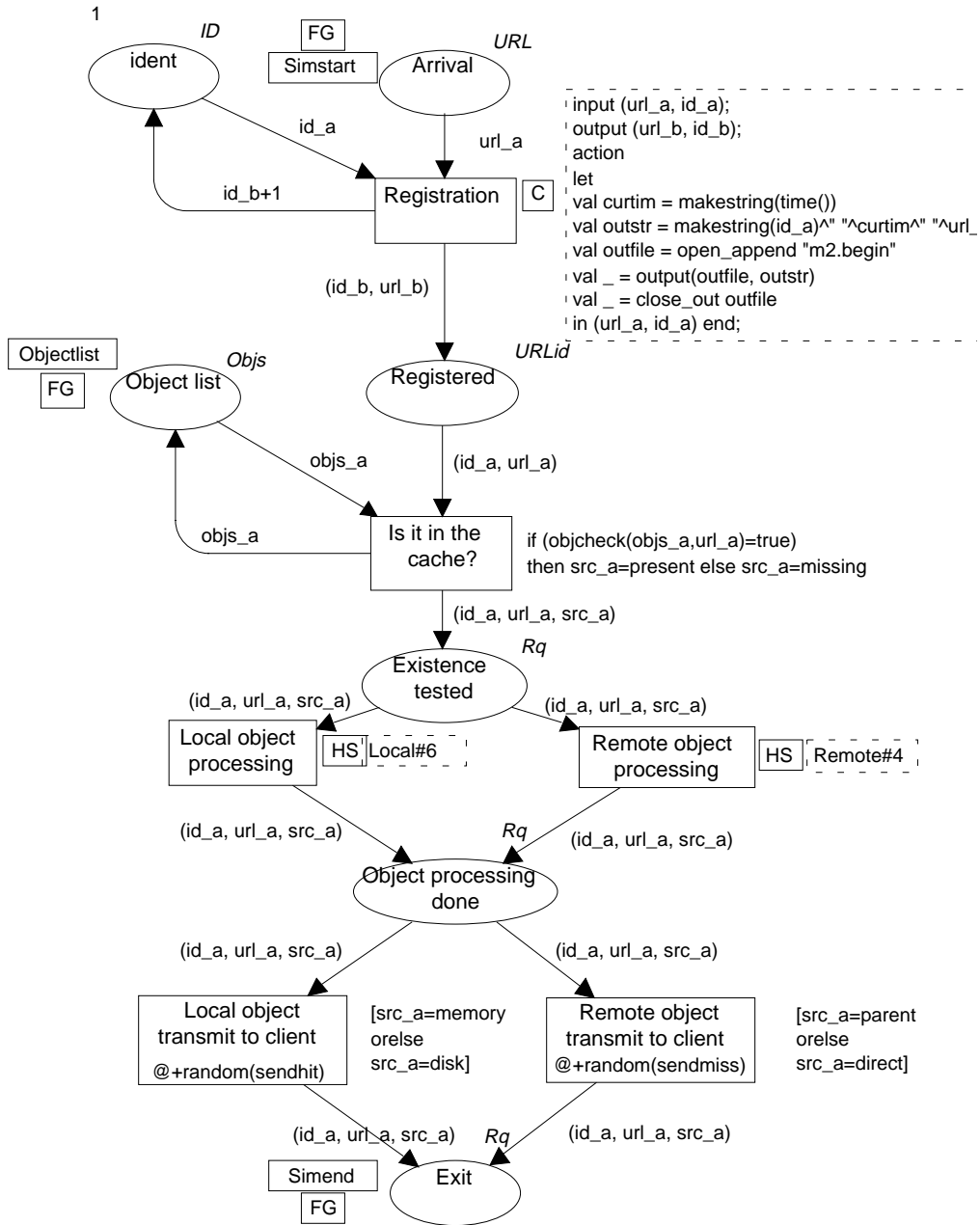


Figure 2: The SUPERPAGE page of the model

3.4 Local object processing: pages LOCAL and MEMORY UPDATE

On this page requests are read from the memory or disc and the last referenced time of the objects is updated.

<i>Local object processing</i>	the requested object is present in the cache
Local object	local request
<i>Locate object</i>	check if the request is in the memory or on disc
Located first	request marked with memory or disc
<i>Disk I/O</i>	input of the object from disc (its time stamp is incremented by a random delay time based on measured data)
Memory update needed	we need to store objects read from disc in memory
<i>Memory updated</i>	memory has been updated (last reference time of the object is updated). If there is no more space in memory then an object is deleted from memory based on last referenced time (see in details on page MEMORY UPDATE).
Memory objects	the list of objects stored in memory manipulated by the branch coming from <i>Memory updated</i>
Update enabled	the object list can be updated after memory update
<i>Read from memory</i>	the object is in memory, we read it and update memory
Memory objects	list of objects in memory updated by the branch coming from <i>Read from memory</i>
Local object	local object processing done

3.5 Remote object processing: page REMOTE

The processing steps when the request is not in the cache are collected on this page. The object is downloaded from the hierarchy or directly from the source. Thereafter the object is stored both in the memory and on the disc marked with the actual system time as its last referenced time. The garbage collection from disc is also performed here based on the last referenced time of the objects.

<i>Is the object in the cache?</i>	the object is not in the cache
Remote object needed	remote processing is needed
<i>Request broadcast</i>	send request: a random variable on the output arc of this transition describes if the response comes from the

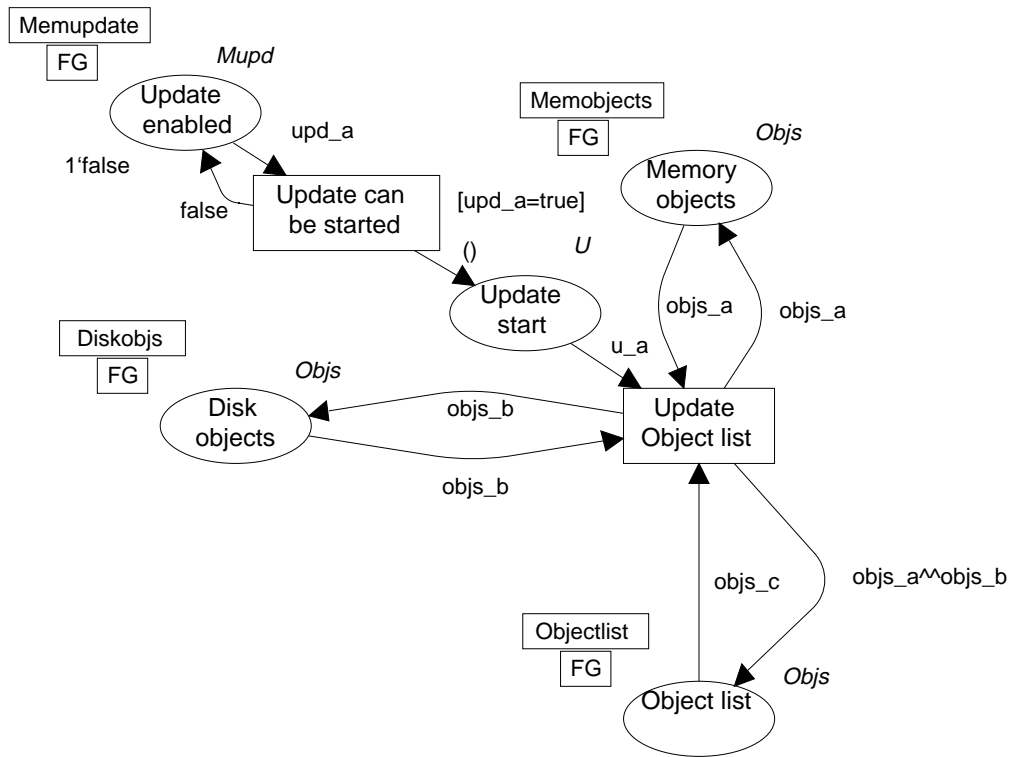


Figure 4: The UPDATE page of the model

	parent or we have to download the object directly from its source
Object or timeout	object to be downloaded marked by its source (directly or from parent)
<i>Timeout, direct connection</i>	the object is downloaded directly from its source, its timestamp is incremented by a constant (tunable parameter) time-out value and a random time associated to the direct download
<i>Object comes from hierarchy</i>	the object is downloaded from the hierarchy (i.e. from the parent), its timestamp is incremented by a random value chosen from a multiset representing the download time from the hierarchy
Remote object received	the object has been received
<i>Update memory</i>	update the content of the memory
Disk update follows	in parallel with the transmission to the client we have

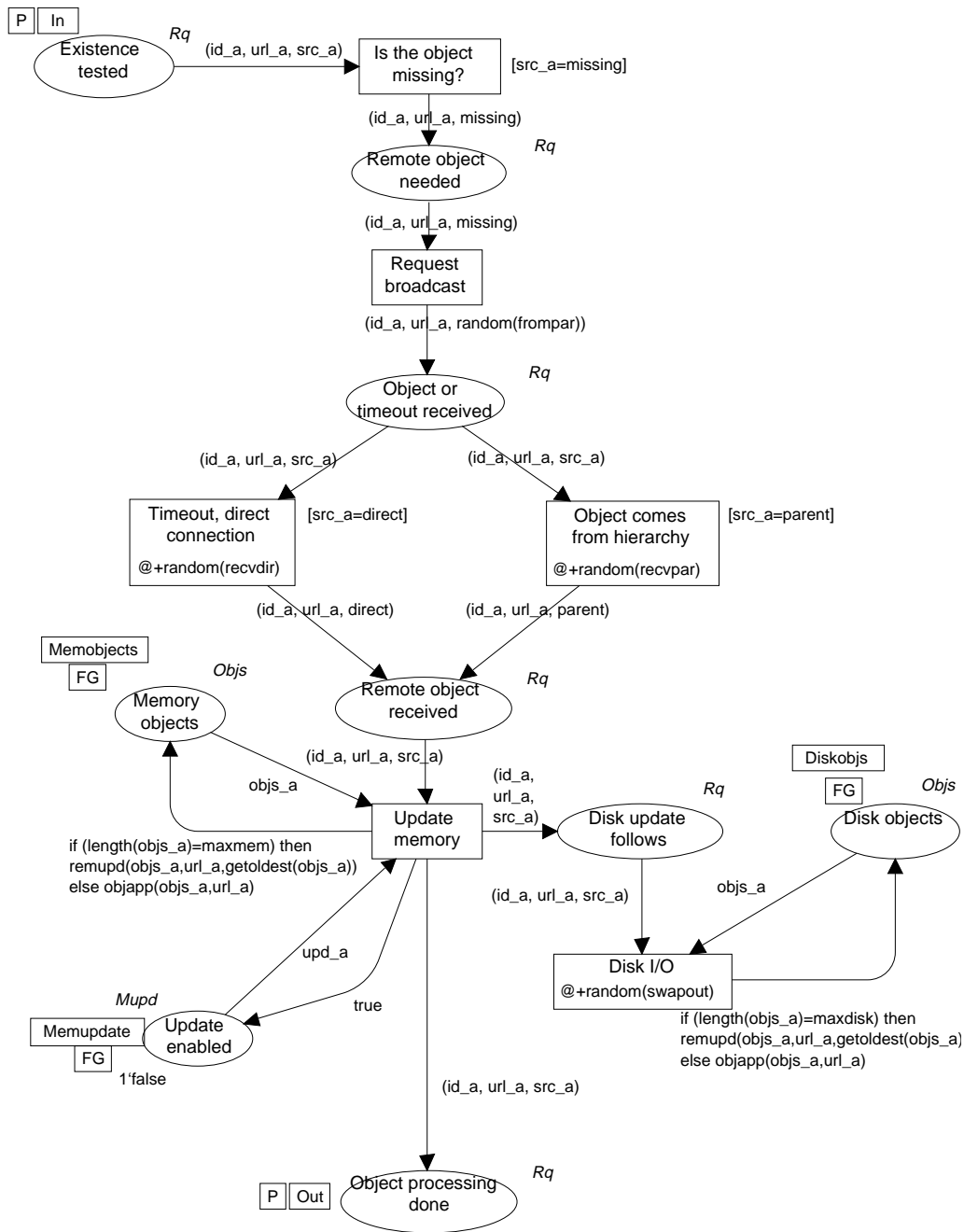


Figure 5: The REMOTE page of the model

to write the object to the disc, too

3.6 The most important tokens present in the CPN model

3.6.1 Tokens associated to requests (color Rq)

Requests are described by special coloured tokens where their colour is a vector of colours with elements

- request identifier: `id` (unique identification number)
- object identifier: `url` (Uniform Resource Locator)
- response source: `src` (local/from parent/from neighbour)

Moreover the requests possess their `time stamp`.

3.6.2 Tokens associated to stored objects (color Objs)

These are color set type tokens present on the places **Memory objects**, **Disk objects** and **Object list**. The color `Objs` is a list of records where an `url` and a `time stamp` showing the last usage time are put into a record.

The detailed description of all the other tokens used and all the procedures written is found in the **Appendix: The contents of the declaration node**.

3.7 Measurable variables and parameters of the model

3.7.1 Variables

The most important dynamic variables described in Section 2.3 are represented in the CPN model above in the following way.

1. *disturbance variables*

The cache load is modelled through the time dependent set of the request tokens on the place **Arrival**.

2. *output variables*

Both the response time and the hit-rate as the main characteristic variables are taken into account. The response time as a distribution is computed from the `time stamp` of the arrived request tokens on the place **Exit** while the hit-rate can be obtained from the `src` color item.

3. *state variables*

The state variables of the cache system are represented by the list of the cached objects on the places **Object list**, **Memory objects** and **Disk objects**.

4. *potential input variables*

The input variables are mainly present in various constraints, i.e. upper limit on available resources in the CPN model, such as

- the maximal number of objects on the place **Object list**,
- the time-out value associated to the transition *Request received*,
- the maximal number of objects on the place **Memory objects**,
- the maximal number of objects on the place **Disk objects**.

3.7.2 Parameters

In addition to the dynamic variables the CPN model contains *parameters* which characterise the modelled cache system. The model parameters are mostly response times of various kind associated to the relevant transitions in the CPN and can also be measured. These parameters possess random and slowly time varying nature, i.e. they can be characterised by slowly time varying distributions. The following table summarises the model parameters and their location in the CPN model.

<i>Identifier</i>	<i>Meaning</i>	<i>Transition</i>
disk-io	disk i/o response time	<i>Disk I/O</i>
trans-resp	transmission time from the from hierarchy	<i>Object comes from hierarchy</i>
send-client	transmission time to the client	<i>Transmit to client</i>

4 Simulation using the CPN model

4.1 Simulation input

A simulation run is performed by specifying the initial marking which represent the following cache variables:

- a set of request tokens put on the place **Arrival**,
- a set of tokens put on the places **Object list**, **Memory objects** and **Disk objects**.

The set of request tokens is placed in an input file generated from the log files of the caching software and read to the CPN simulator.

4.2 Simulation output

As a result of the completed simulation

- a set of processed request tokens on the place **Exit** and
- a set of new tokens on the places **Object list**, **Memory objects** and **Disk objects**

is generated.

Since we are primarily interested in distribution of the processing time of each request, the tokens collected at the place **Exit** are written out to an output file for further processing.

4.3 Simulation results

All the branches have been tested individually using specially designed simulation inputs and observing the simulation output. We do not present any slides from the simulation run, because the large number of tokens makes it impossible to follow the simulation on paper.

5 Model verification

To be able to eliminate irrelevant details from our model and to verify it against real behaviour we need to measure the static and dynamic characteristics of the modelled object cache. The dynamic variables (input, state, output and disturbance variables) of our CPN model have been measured using special tools. In addition, the model parameters above have also been determined from measurements either directly or in an indirect way.

5.1 Measurements

The caching software used at the modelled cache was Squid-1.1.17. [8] This software generates two log files which are useful for analysing the cache activity. We applied a patch made available by Alex Rousskov [9] which enhances the log file records by some time profiling data. It must be stressed that these measured data contain per request based measurements since the profiling data are measured by the caching software for every request. Details on the measurements and measured data have been reported elsewhere [4].

The model parameters have also been determined from these measured data.

- *Disk input/output*: the distribution of time needed for swapping an object from/to the disk to/from the virtual memory
- *Processing time of local object*: the distribution of time needed for the service of a local object (i.e. an object that can be serviced from the cache)
- *Retrieval time from the hierarchy*: the distribution of time needed for object retrieval from a different cache
- *Retrieval time from the source server*: the distribution of time needed for object retrieval from the source server
- *Request arrival rate*: the distribution of time between successive client requests

5.2 Comparison of the simulated model output and the measured output variables

The initial marking, i.e. the simulation input of a simulation run have been generated from the measured and evaluated per request based log files in different cases, i.e. for different cache load and time interval. The model output is generated therefrom by performing a simulation run and collecting the simulation output.

The model output is then compared to the records of the same log files the simulation input was generated from. The comparison is shown on Fig. 5.2. We can say that good correspondence has been found between the measured and simulated data based on our log files. The shape of the two data sets shows identical nature. The difference between the simulated and the measured data originates from *the gross tuning parameters*: we used a very limited number of data to specify the distribution of the model parameters. (The specification of model parameters can be seen in the Appendix).

6 Conclusion and Future work

A novel stochastic timed CPN model of an Internet object leaf cache has been developed and verified in the paper. The model has been implemented and tested in the Design/CPN software environment.

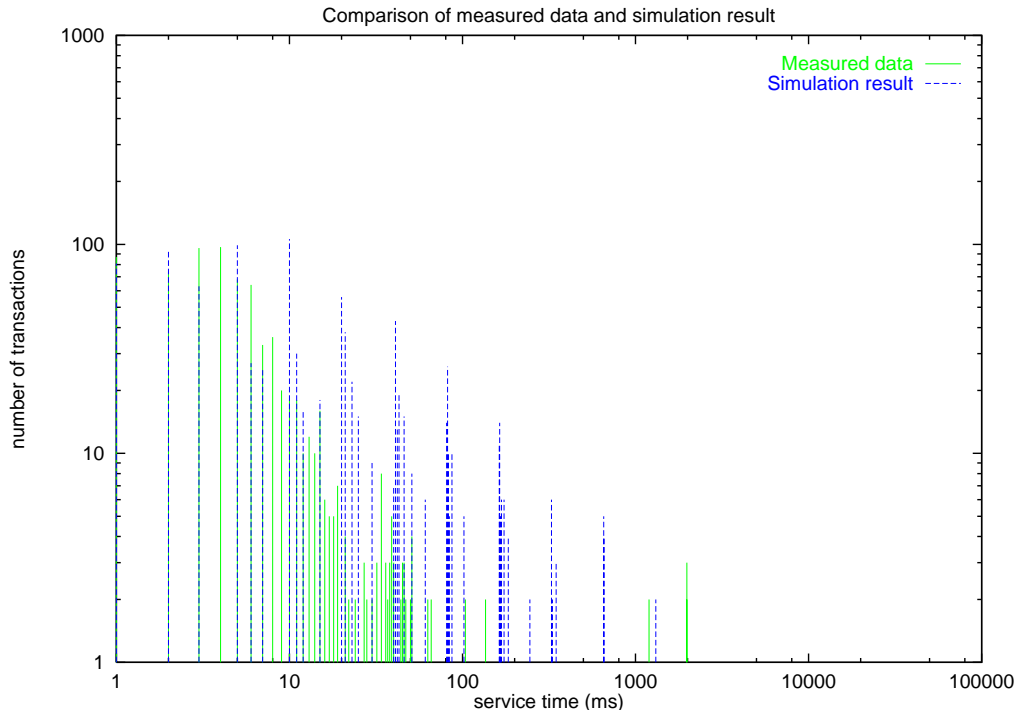


Figure 6: Comparison of measured data and simulation result

The aim of the modelling has been to use the model for control and optimization purposes: to develop a model-based adaptive methodology for tuning the cache parameters based on measured data and on the desired objective (loss) function of the cache performance.

Having verified the CPN model the following model validation steps are needed which will be the subject of our further research.

1. validation of the model output against measured data in different cache load regimes,
2. model sensitivity analysis with respect to the variations in model parameters,
3. model sensitivity analysis with respect to the variations of the potential input variables,
4. model refinement and simplification according to the above,
5. adaptive tuning experiments.

Acknowledgement

This work has been partially supported by the Hungarian National Research Fund (OTKA) through grant T026575 which is gratefully acknowledged.

References

- [1] Evangelos P. Markatos: Main memory caching of Web documents. In *Proceedings of the Fifth International World-Wide Web Conference*, Paris, France, May 1996.
- [2] Roland P. Wooster, Marc Abrams: Proxy caching that estimates page load delays. In *Proceedings of the Sixth International World-Wide Web Conference*, Santa Clara, California - USA, April 1997
- [3] B. Kolics: Toward Intelligent Internet Object Caching (Diploma thesis). University of Veszprém, 1997.
- [4] B. Kolics, K. M. Hangos, I. Tétényi: Experimental Investigation of Internet Object Cache Characteristics, *INET'98 Conference*, July 21–24, Genève, Switzerland
- [5] Ho, Y. C., Cassandras, C.: A New Approach to the Analysis of Discrete Event Dynamic Systems. *Automatica*, **19**, 149–167 (1983)
- [6] Kurt Jensen: Coloured Petri Nets, *Basic Concepts, Analysis Methods and Practical Use*, Vol. 1–3, Springer Verlag, 1992-1997
- [7] Design/CPN Online home page:
<http://www.daimi.aau.dk/designCPN/>
- [8] Squid Internet Object Cache home page: <http://squid.nlanr.net/>
- [9] Squid Profiling Statistics home page:
<http://www.cs.ndsu.nodak.edu/~rousskov/research/cache/squid/profiling/>

Appendix: The contents of the declaration node

```
color ID = int; (* request identifier *)
color URL = string declare input_ms timed; (* object identifier *)
color Lrt = int; (* last referenced time *)
color Rt = with simple|conditional; (* request type *)
color Src = with present|missing|memory|disk|parent|neighbour|direct;
(* object source *)
color Obj = product URL * Lrt declare input_ms; (* object string
*)
color Objs = list Obj; (* object string list *)
color URLs = list URL; (* URL list *)
color URLid = product ID * URL timed;
color Rq = product ID * URL * Src timed declare output_ms; (* request
record *)
color Lrq = product ID * URL * Src * Lrt timed; (* local request
record *)
color U = unit; (* unit for initialization *)
color Mupd = bool; (* used for memory update enabling/disabling *)

var id_a,id_b: ID;
var url_a, url_b: URL;
var urls_a: URLs;
var urlms: URL ms; (* URL multiset variable *)
var objs_a, objs_b, objs_c: Objs;
var src_a, src_b: Src;
var lrt_a: Lrt;
var upd_a: Mupd;
var u_a: U;
var rq_a: Rq;

val maxmem = 1000; (* maximum number of objects in memory *)
val maxdisk = 100000; (* maximum number of objects on disk *)
val reqarrival =4'1+804'2+758'4+532'8+628'16+1119'32+1771'64+3164'128+
3938'256+3056'512+2921'1024+2109'2048+1117'4096+791'8192+389'16384+
142'32768+18'65536+4'131072+2'262144+90'524288;
(* time delay between successive requests *)
val swapout = 17'4+2199'8+3933'16+17742'32+35209'64+7807'128+3118'256+
908'512+266'1024+203'2048+75'4096+3'8192+1'32768;
(* swapout time *)
```

```

val swapin = 17'2+6246'4+4115'8+4724'16+8076'32+11239'64+6753'128+3143'256+
1222'512+312'1024+129'2048+32'4096+5'8192+1'16384+1'32768;
(* swapin time *)
val sendhit = 12'2+4168'4+11007'8+6074'16+5061'32+7470'64+6290'128+3868'256
+1734'512+574'1024+330'2048+139'4096+76'8192+69'16384+57'32768+
58'65536+18'131072+19'262144+7'524288+1'1048576;
(* time it takes to process a 'hit' object *)
val sendmiss = 9'1+1068'2+16224'4+19586'8+4807'16+6159'32+8617'64+
11361'128+9841'256+6904'512+4796'1024+4674'2048+3322'4096+2113'8192+
1124'16384+623'32768+363'65536+170'131072+79'262144+57'524288+
33'1048576+25'2097152;
(* time it takes to process a 'miss' object *)
val recvpar = 3'4+101'8+155'16+592'32+1500'64+2888'128+3762'256+4834'512
+3238'1024+2609'2048+2393'4096+1425'8192+672'16384+287'32768+197'65536+
87'131072+35'262144+32'524288+19'1048576+9'2097152;
(* hierarchy response time *)
val recvdir = 21'8+360'16+257'32+618'64+1184'128+1584'256+1515'512+
1680'1024+1136'2048+841'4096+575'8192+351'16384+132'32768+60'65536+
21'131072+4'262144+2'524288;
(* direct object retrieval time *)
val frompar = 293'direct+607'parent;
(* relation of direct and hierarchy responses *)
(* derived from log file analysis *)

fun getmin(q: Obj, p: Obj):Obj = if (#2(q)<#2(p)) then q else p;
(* returns the object with the lower *)
(* last referenced time value *)

fun last_used(l: Objs, a: Obj, i: int):Objs*Obj*int = if (i=length(l))
then (l,a,i) else last_used(l,getmin(a,nth(l,i)),i+1);
(* returns the object string which has the *)
(* lowest last referenced value in the list *)
(* in the second field *)

fun remove_last_used(l: Objs):Objs =
ms_to_list(list_to_ms(l)-1'#2(last_used(l, (" ",maxage),startpoint)));
(* removes the last referenced object string *)
(* from an object string list *)

fun objcheck(objlst, myurl) = exists (fn elm => elm = (myurl, #2(elm)))
objlst;

```

```

(* tests whether an URL is in the list *)

fun updateobj(objlist, myurl) = map (fn elm => if (myurl=#1(elm))
then (myurl, time()) else elm) objlist; (* updates the timestamp
of an URL in a list *)

fun remupd(objlist, myurl, oldurl) = map (fn elm => if (oldurl=#1(elm))
then (myurl, time()) else elm) objlist; (* removes oldurl from list
and inserts myurl *)
(* to list with actual timestamp *)

fun geto3(l:Objs, i:int, j:int):int = if (#2(nth(l,i))<#2(nth(l,j)))
then i else j;
(* assistant function of getoldest *)
fun geto2(l:Objs, i:int, j:int):Objs*int*int = if (i=length(l)) then
(l,i,j) else geto2(l,i+1,geto3(l,i,j)); (* assistant function of
getoldest *)

fun getoldest(l:Objs):URL = #1(nth(l,#3(geto2(l,0,0))));
(* returns the URL that has the lowest LRT value *)

fun objapp(l:Objs, u:URL):Objs = l^[u,time()];

```