A bi-objective Mixed Integer Linear Program for load balancing DNS(SEC) queries

Stanislas Francfort, Daniel Migault, Stéphane Sénécal

Orange Labs
38-40 rue du Général Leclerc, 92794 Issy-les-Moulineaux CEDEX 9, FRANCE
{stanislas.francfort, daniel.migault,
stephane.senecal}@orange-ftgroup.com

Abstract. This paper addresses the problem of efficiently load balancing domain name queries on DNS resolution platform. Efficiently means that resource required for the resolution MUST be optimized compared to existing architectures and equally shared among the nodes of the resolving platform. This optimization leads to well balance the load, which reduces drastically the number of servers to be deployed as well as to significantly save power consumption. In order to achieve this goal, the paper considers splitting DNS traffic according to the queried FQDN – rather than IP addresses. Then, based on the specific statistical distribution of the DNS queries, we define an efficient method so that nodes deal with the roughly same number of FQDN, as well as to deal with the same number of DNS queries. We formulate this bi-objective problem under the framework of a Mixed Integer Linear Programming model and use a solver to process the resulting optimization problem. The model we developped is very efficient on real data, and provides a promissing offline process for load balancing DNS queries on a large resolution platform.

Keywords: DNS, DNSSEC, load balancer, Integer Programming.

1 Introduction

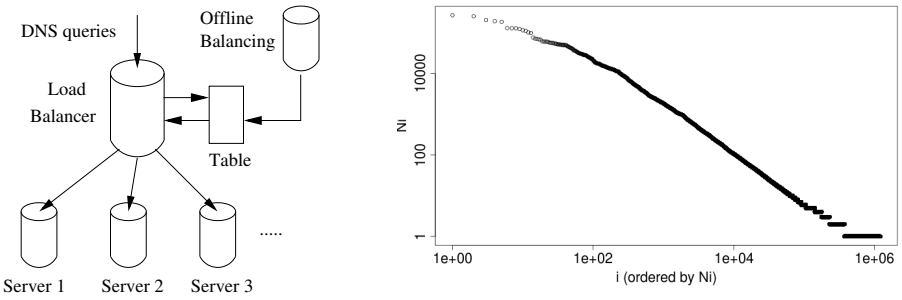
Domain Name System (DNS) [1, 2] is the protocol used to bind a Fully Qualified Domain Name (FQDN) like *www.google.com* to an IP address. Thus every time end users are typing an URL in their web browsers, a DNS resolution is performed in order to find where the web server is located.
Today, DNS resolving servers deal with traffic that has a daily mean of 40 000 queries per second, with flash crowds up to 120 000 which corresponds to the DNS traffic Orange has with its residential End Users. On the other hand, the DNS traffic keeps on increasing and roughly double each year. DNS has been designed in the eighties so that resolution could be efficient and involve the least possible resources. As such, the DNS resolving platforms are usually splitting traffic between the servers, thanks to load balancers that only consider networks layers, i.e. IP addresses. As a result, the traffic is uniformly split among the servers, and each server deals with the same amount of queries. The advantage

of such architecture is that it is very scalable, and when the traffic increases, network administrators only need to add more servers. The drawback of this architecture is that there is no synchronization between the servers' caches and that servers perform parallel DNS resolutions for popular FQDN. This has not been an issue with DNS since resolutions are quite straightforward. However, with DNSSEC, the DNS SECurity extension, cf. [3–5], resolution requires one or more signature checks as well as longer datagrams. In fact such resolutions require much more resources than DNS's resolution and [6, 7] show that, depending on the software implementation, the platform requires between 2 and 4.25 times more resource with DNSSEC than with DNS.
A key point is to observe that if the same DNSSEC query has been addressed previously to the same server within a given period of time, defined by the variable "Time To leave" (TTL), the server does not perform another resolution over internet and sends back a response already stored in cache. In order to save computational resources, we aim at optimizing the number of queries for which the response is stored in cache. By affecting every query of a given FQDN to the same server (or node), we achieve for each FQDN only one cryptographic resolution every TTL seconds. On the other hand traditional architectures that splits DNS traffic without considering the FQDN, can perform up to J cryptographic resolutions every TTL where J is the number of nodes of the resolving platform.
In our solution, the distribution, that is to say the definition of which FQDN goes on which server is defined by an offline process (Fig. 1a). Once computed on a captured data, the table is stored and used to load balance the queries online. The table has to be computed for each given period of time. This split is performed according to the requested FQDN and does not consider the network layer parameters such as IP addresses. In this case, each server of the platform is responsible for a set of FQDN, and all queries for a given FQDN will be resolved by a specific server. The matching FQDN/Server is stored in the table. This maximizes the probability that the response is already cached, however, by doing so, there is no guarantee that servers will have to deal with an equal number of queries.
In this paper, we are looking how the traffic can be distributed so that the two following criteria can be satified:

1. The number of FQDN is well balanced between the different servers of the platform.
2. The number of DNS queries is well balanced between the different servers of the platform.

The DNS traffic considered in this paper is 5 min DNS capture from our residential End Users at the rush hour $19h33 - 19h38$ on October 19 2009.
Criterion 1 ensures that each server will store approximately the same amount of data in its cache, aiming to perform approximately the same number of cryptographic computations while criterion 2 ensures that each server resolves approximately the same number of queries. Optimizing along those two objective functions turns the problem into a bi-objective well-balancing integer program

on a huge data set captured from real DNS traffic.
As far as we know, no such study on minimizing cryptographic DNSSEC computations by optimizing the queries to the cache have been conducted, nor any solutions addressing the large scale problem of well balancing a huge number of integers into a small number of boxes[1]. Section 2 introduces the problem mod-



(a) DNSSEC Load Balancer (b) Repartition of N_i (log-log)

Fig. 1. Platform & Traffic

eled as a Mixed Integer Linear Program (MIP). We also point out the fact that the size of the instances is so huge that it can not be solved in such a way. In section 3 we explain how the very specific statistical distribution of the DNS queries allows us to split the data into two parts, one solved with an efficient MIP, the other balanced with a fast heuristic. Experimental results, and comparisons to existing solutions such as round-robin, showing the efficiency of our solution, are presented in section 4. Finally, section 5 concludes the paper and describes future work.

2 Modeling the problem as a single objective Mixed Integer Linear Program

The problem ($\mathcal{P}$) we have to solve, in order to fill the table, can be formulated as follows: "minimizing the difference of the number of queries received by each server while minimizing the number of FQDN received by each servers with respect to the constraint: each query of a given FQDN has to be sent to the same server".

[1] Note that this problem is different from the Bin-Packing problem or the Knapsack problem. Our problem could be reduced to those ones if we had to fit exactly the same amount of queries (resp. signature checks) to each server.

2.1 Aggregating the two objectives

First, note that this is a bi-objective problem (i.e. which optimizes along two objective functions), and in order to manage it, we construct a single aggregate objective function.
We aggregate them in a standard way, by a linear combination of the two objective functions. Let $k \in \mathbb{R}$ be an aggregating parameter. In order to compare the respective computing resources of a unique standard DNS query C_{req} and a unique cryptographic signature check C_{sig}, we can define k to be $\frac{C_{\text{sig}}}{C_{\text{req}}}$. This aggregation is modelized page 5 by inequation (9) and objective function (10).
[9] shows that if the checks of the signature computation are stored in cache, than the servers can be loaded by 3.33 times more queries than if the checks are not in cache.
This leads us to a good approximation of $\frac{C_{\text{sig}}}{C_{\text{req}}}$ and we will set the aggregating parameter k to be equal to 3.33 in the remaining of the paper. However as [7] shows such values may vary with the used implementation and its configuration.

2.2 Modeling the problem as a Mixed Integer Linear Program

In the following we denote by $i \in I$ the FQDN, and by N_i the number of queries whose object is FQDN i.
In a previous work, we modeled the problem into four different MIP models, and we compared them. One of them performed far more efficiently than the three others. We present the four models in appendix as well as the comparison of how fast they converge. In the present paper, we only present the most efficient one, and we will refer to it as "the" MIP model. Note that this result is conform to the studies we can find in [13].
Consider the following constraints and variables:

$$I \quad \text{set of FQDN} \tag{1}$$

$$J \quad \text{set of servers} \tag{2}$$

$$x_{i,j} = \begin{cases} 1 \text{ if FQDN } i \text{ is assigned to server } j \\ 0 \text{ else} \end{cases} \tag{3}$$

$$k \in \mathbb{R} \quad \text{an aggregating parameter} \tag{4}$$

$$N_{i \in I} \quad \text{the number of query for FQDN } i \tag{5}$$

We can now introduce the following variables: $\forall j \in J$ a server, let S_j (resp. T_j) be the total amount of queries (resp. FQDN) for server j.
It is then possible to express problem $(\mathcal{P})$ as a MIP with 3 sets of constraints

and a single objective function:

$$S_{j\in J} = \sum_{i\in I} N_i \times x_{i,j} \quad \forall j \in J \tag{6}$$

$$T_{j\in J} = \sum_{i\in I} x_{i,j} \quad \forall j \in J \tag{7}$$

$$\sum_{j\in J} x_{i,j} \geq 1 \quad \forall i \in I \tag{8}$$

$$S_{j_1} + k * T_{j_2} \leq M \; \forall j_1, j_2 \in J \tag{9}$$

$$\text{Objective function: minimize } M \tag{10}$$

3 DNS queries data

In this section we show that our MIP model does not behave very well on real data. Actually, it is not possible to solve a MIP on such a huge data. But thanks to the specific statistical distribution of the data, we are going to derive another far more efficient model.
In order to guarantee that data for this study are realistic, we considered DNS queries captures from operational DNS servers of a telecommunication operator company measured in 2009. The main file is a brute capture of 5 mn of typical traffic, which represents about 800 MB of raw data.

3.1 DNS statistical distribution

The table 2a lists the ten most required FQDN during the 5 minutes of capture of the network traffic flow. Table 2a clearly shows that the 10^{th} requested FQDN is half less popular then the most popular FQDN. Indeed, the DNS queries repartition is very imbalanced. Thus, within 5 mn, we captured 17 299 154 queries distributed in 1 211 880 FQDN which leads to an average of 14.27 queries per FQDN.
However, this average value is not very meaningful. In fact, the minimum of this repartition is 1 query, but the median is also equal to 1.
There are actually 837 154 FQDN requested a single time and 374 726 FQDN requested strictly more than once. As a counterpart, the maximum values 271 586 and is obtained for an unique FQDN.
This observation on the data points the fact that the statistical distribution is very specific and could be used to derive a more efficient model than solving a MIP on the whole data. Let us study in details this very imbalanced repartition in the remaining of the section and take it into account in order to decompose our problem into two subproblems. From [9] which studies clustering DNS data, it is noticeable that the different FQDN have quite different query frequencies, which are distributed like a power law. In fact, some phenomena in which the notoriety increases because of notoriety (Matthew Effect) give birth to power law

name	nbr_occur
www.facebook.com	271586
wpad	256144
arpa	206656
ad.fr.doubleclick.net	193632
www.google.com	187028
view.atdmt.com	131181
www.yahoo.com	130095
profile.ak.fbcdn.net	129737
www.google-analytics.com	124445
www.google.fr	116170

(a) Most frequent FQDN

Q	0%	25%	50%	75%	100%
Q-quantile	1	1	1	2	271 586

(b) Quantiles of the Cumulative Distribution Function of the N_i's

Fig. 2. FQDN Traffic Repartition

distribution. [10] describes an analysis of the queries distribution with respect to domain names on the AOL servers during one day. The conclusion deduced from the analysis is that the repartition of queries with respect to domain is indeed distributed as a power law distribution. The case of popularity in DNS analysis is similar.

However, we do not have to prove that the repartition of FQDN queries follows a power law distribution. Instead, our interest is to investigate the global behavior and to note that we can estimate it roughly by a power law. Under the power law, there is a great imbalance between the most frequent events and the rare ones, while the rare events are numerous. This observation allows us to design an efficient method to resolve the problem.

The Fig. 1b represents the distribution of N_i from the most frequent to the rarest. Depicted in the graph with a log-log scale, this distribution is roughly linear, which is one of the properties of power law distributions.

The quantiles of this distribution are shown in Table 2b. They are interesting for demonstrating the imbalance characteristics of queries repartition N_i among the different FQDN i.

The frequencies associated to FQDN definitely follows a largely non-Gaussian distribution law[2]. These characteristics make uniform repartition among the servers of the platform a very inefficient way to split the DNS traffic. Simulations confirmed this fact as we can see in the first line of table 3. Indeed, the query rate associated to each FQDN is very imbalanced, then, affecting too many popular FQDN to a single server would result in a very imbalanced distribution of the incoming traffic. The following section explain why the proposed solution works well if the data is distributed as a power law.

[2] However, claiming without ambiguity this is indeed a power law would require more than the arguments presented in this paper, as shown in [11, 12].

3.2 Decomposition into two subproblems

In order to make DNSSEC traffic well balanced among the different servers, a strategy resulting from this very imbalance distribution is to share the problem into two subproblems, i.e. defining the set of FQDN whose distribution is defined using MIP and the set of FQDN whose distribution can be defined with a uniform stateless function like a hash function.
Let s be a threshold, and I_s the set of FQDN receiving more than s queries. From now on, we are no longer trying to calibrate s, but instead, the linked number $\|I_s\|$, which is the one we want to choose in order to separate efficiently the two subproblems.
The first subproblem is to assign carefully the $\|I_s\|$ FQDN that are frequently requested to servers, by solving the MIP on this subset of FQDN.
The second one is to assign uniformly the $\|I - I_s\|$ others. Observe that since DNS queries are temporally uncorrelated, this uniform repartition is similar to a round-robin repartition on servers.
We are clearly taking advantage of the imbalanced distribution.
Table 3 shows indeed that the number of queries managed by solving a MIP increases very fast when $\|I_s\|$ increases, thanks to the very imbalance distribution of N_i. Moreover, simulations on real data shows that when a uniform distribution is performed over the remaining $\|I - I_s\|$ FQDN, the imbalance due to those FQDN not performed by the MIP decreases very fast. Let us call "residual imbalance" this imbalance, calculated as the difference between the load of most loaded server and the load of the less loaded one. The third column in the table 3 shows how this residual imbalance decreases very fast with respect to $\|I_s\|$.
The portion of the most frequently requested FQDN we have to consider in

$\|I_s\|$	$\sum_{N_i>s} N_i$	Residual imbalance $S_{\max} - S_{\min}$
1	1	805 329
100	5 524 529	233 180
200	7 043 815	141 988
300	8 030 281	111 583
400	8 656 026	88 818
500	9 105 475	77 275
700	9 734 240	63 588

(a) 1 - 700 FQDNs

$\|I_s\|$	$\sum_{N_i>s} N_i$	Residual imbalance $S_{\max} - S_{\min}$
1 000	10 390 719	50 768
2 000	11 623 826	27 468
4 000	12 683 199	15 579
8 000	13 562 536	10 414
16 000	14 280 865	6 202
32 000	14 885 176	3 799
48 583	15 193 650	2 625
374 726	16 462 000	1 009

(b) 1000 - 374726 FQDNs

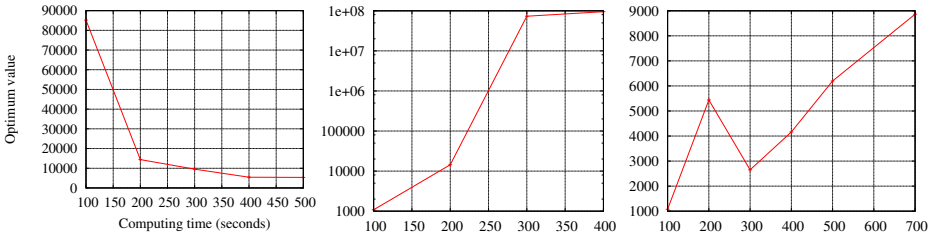
Fig. 3. Number of FQDN managed by MIP and residual imbalance w.r.t. I_s

the MIP results from a trade-off. In fact the more FQDN we consider in the MIP, the better the traffic will be balanced among the servers. However the more FQDN we consider, the bigger the MIP will be, and hence the more time,

the more resources are required. Note that if $\|I_s\|$ increases too much, we can expect the MIP to become intractable. On the contrary, when $\|I_s\|$ decreases, the optimization problem may become easy, but the imbalance increases very fast (cf. Table 3).
The power-law like distribution described in section 3 ensures on one hand that the imbalance decresases very fast, and on the other hand that most of the queries will be affected to servers by the MIP, even if $\|I_s\|$ is very small which makes our solution far more efficient that round-robin. As a consequence, due to the power law like distribution of the data, we can expect to find an $\|I_s\|$ small enough for the MIP to be tractable, but big enough to lead to an acceptable small residual imbalance.
Note that the number of variables of the MIP is $\|I_s\| \times \|J\|$ where $\|I_s\|$ is the number FQDN balanced by the MIP, and $\|J\|$ is the number of servers.
As examples, choosing the threshold s to be the median $= 1$ (resp. the mean $= 14.27$) of the N_i's leads to an optimization problem to solve with 374 726$\times\|J\|$ (resp. 48 583 $\times \|J\|$) variables.
On the other hand, in [9] is described an innovative clustering methodology which leads to 4 clusters where 3 of them are very different from the fourth one. Those 3 clusters are the more frequently asked FQDN found on DNS data. More precisely, from 30 seconds of DNS capture, which gave 167 793 queries the "adaptative k-means" (resp. "k-means") algorithm gave 84 (resp. 143) FQDN in the 3 clusters. In this case, we can choose I_s in order its cardinal $\|I_s\|$ to be 200.
The proposed platform sketched in the introduction can be now presented: to solve the bi-objective load balancing problem on a huge data, this data is split into two subsets and then a repartition on one of the subsets is performed by solving a MIP. This repartition is stored in a table. Once this table is full, the load balancer is able to well balance very efficiently the whole incoming DNS traffic, whether by a call to the table for the most frequent FQDN, or by a uniform repartition for the remaining FQDN.

4 Numerical experiments

Once the model has been defined, we solve it by writing it into a modeling language GNU MathProg and by using the open source GLPKv4 LP-MIP solver. Computations have been conducted on an AMD Athlon II X3 powered by a Linux 2.6 kernel. From now on, we fix the number of servers to be $\|J\| = 10$.
Fig. 4a shows that GLPK converges quite quickly when solving the MIP model on 200 FQDN. We can notice as well that the remaining balance achieved, once past the 200^{th} second, is very small.
Fig. 4b and 4c show how the imbalance increases when the number of FQDN managed by the MIP varies. This imbalance is computed as the difference between the maximum and minimum number of queries sent to the server ($\max(S_{j_1}) - \min(S_{j_2})$ for j_1 and j_2 in J). Computations are respectively done for 200 and 3 600 seconds, which are good trade-off between the number of



(a) MIP convergence (b) Δ w.r.t. $\|I_s\|$ (200 s) (c) Δ w.r.t. $\|I_s\|$ (3 600 s)

Scenario	Computation time	Imbalance	# of sig checks
MIP-200	200	156 386	1 211 880
MIP-700	3 600	72 453	1 211 880
RR-FQDN	0	805 329	1 211 880
RR-Req	0	0	2 493 880

(d) Evaluation of the scenarios

Fig. 4. Performance Evaluation (with $\Delta = S_{\max} - S_{\min}$)

resolutions avoided and computing time to full the table.
In Fig. 4b GLPK has been stopped after 200 seconds. It appears a steep slope between 200 and 300 FQDN. This is the range where the MIP becomes too large to solve for GLPK. On this size of instance, it has not had time enough to branch efficiently. The solution found on instances greater than 300 is so bad than we had to represent the imbalance logarithmically on the figure.
The maximum number of FQDN GLPK can solve on this MIP in less than 200 seconds is 400. Beyond this point, no feasible solution is found.
This leads us to an efficient scenario, called scenario MIP-200, which consist on balancing 200 FQDN by solving a MIP in 200 seconds.
The total imbalance is then the sum of the imbalance due to the MIP and the residual imbalance, that is $14\ 397 + 141\ 988 = 156\ 386$; and the number of signature checks is equal to the number of servers $\|I\| = 1\ 211\ 880$. In this case the difference between the maximum and the minimum number of FQDN sent to the servers is 4.
Fig. 4c shows the imbalance when the size of the problem varies. GLPK has been stopped after 3 600 seconds. It appears a peak at 200 FQDN. This unexpected behavior shows how difficult it is for GLPK to solve the problem when there is only the most frequent FQDN to manage; very large and very different numbers are hard to balance. However, when the number of FQDN grows, the number of small N_i grows as well, and the problem becomes easier to solve as it is easier to fill differences with small numbers.
The maximum number of FQDN that GLPK is able to handle efficiently on this MIP in less than 3 600 seconds is 700. This leads us to the scenario MIP-700, which consist on balancing 700 FQDN by solving a MIP in 3600 seconds.

The total imbalance is then the sum of the imbalance due to the MIP and the residual imbalance, that is $8\ 865 + 63\ 588 = 72\ 453$; and the number of signature checks is equal to the number of servers $\|I\| = 1\ 211\ 880$. In this case, the difference between the number of FQDN assigned to the server overloaded and the server underloaded is 19, which is very few compared to the size of the problem.
In order to get an evaluation of the gains provided by scenarios MIP-200 and MIP-700, we can compare them to two other scenarios, namely scenarios RR-FQDN and RR-Req.
Scenario RR-FQDN (round-robin on FQDN): each query of a given FQDN is sent to a unique server, and the set of FQDN is uniformly distributed among the servers, without considering the frequency associated to each FQDN. This uniform distribution of the FQDN on the servers can be achieved with a round-robin.
In scenario RR-Req (round-robin on queries): DNS queries are uniformly distributed among the servers without considering FQDN. This solution is the classic DNS solution as well as the way load balancing is performed in practical DNSSEC architectures.
Table 4d summaries the results. Solving a MIP on 200 to 3 600 seconds or performing a round-robin on FQDN can save half of the number of cryptographic computations (and corresponding power consumption) involved in the DNSSEC protocol than with the classic DNS solution RR-Req. Note as well that MIP-200 (resp. MIP-700) divide by 5 (resp. 11) the difference between the less loaded server and the most loaded one compares to RR-FQDN. This means that the platforms could be designed for a lower charge.
In other words, if we consider that the bottleneck of DNSSEC migration is due to CPU, and as shown by [6], that migration to DNSSEC requires between 2 and 4.25 times the number of servers, then MIP based distribution requires from 62% to 75% of servers.

5 Conclusion

DNS resolving platforms that are looking to optimize their cache needs to split the traffic between the different servers by considering the FQDN of the queries. By doing so we need to find a way so that the traffic as well as the resources required for the resolution is well balanced between the different servers.
This paper shows how to balance the DNS traffic in a platform composed of multiple servers. The traffic is considered as balanced if every server performs the same number of signature checks and furthermore, if servers also have to deal with the same amount of incoming traffic.
The paper solves this problem by splitting the data into two subsets and then performs a repartition on one of the subsets by solving a MIP. This repartition is stored in a table. Once this table is set, the load balancer redirects any DNS query to the defined server by looking into this table. If the FQDN is not in the table, then a uniform repartition is used like a hash function performed on the FQDN, or a round-robin.

Based on real DNS capture, we show that this solution can be deployed and improve the platform's performance, that reduces by up to 75% the platform's number of servers needed compared to traditional existing platforms.
In order to deploy an operational solution, it could be interesting to investigate more specifically how the frequency of DNS queries is varying over time. This would give an estimation of the latency we should wait before to re-run the MIP on a data set in order to recalibrate it.
Another perspective would be to improve the models and to study the possibilities of combinatorial algorithms taking into account the power law like repartition of DNS queries in order to optimize more efficiently and thus reduce computation resources needed to administrate DNSSEC platforms.

References

1. P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987.
2. P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987.
3. R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005.
4. R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005.
5. R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035 (Proposed Standard), March 2005.
6. D. Migault. Performance Measurements With DNS/DNSSEC. IEPG/IETF79, November 2010.
7. D. Migault, C. Girard, and M. Laurent. A Performance view on DNSSEC migration, November 2010.
8. J.K. Karlof. *Integer programming: theory and practice*. Operations research series. Taylor & Francis/CRC Press, 2006.
9. Q. Xu, D. Migault, S. Sénécal, and S. Francfort. K-means and adaptative k-means algorithms for clustering DNS traffic. Valuetools Conference on Performance Evaluation Methodology and Tools, 2011.
10. L.A. Adamic and B.A. Huberman. The Nature of Markets in the World Wide Web. *Quaterly Journal of Electronic Commerce*, 1:512, 2000.
11. M.L. Goldstein, S.A. Morris, and G.G. Yen. Problems with Fitting to the Power-Law Distribution. *The European Physical Journal B - Condensed Matter and Complex Systems*, 41(2):255–258, 2004.
12. A. Clauset, C.R. Shalizi, and M.E.J. Newman. Power-Law Distribution in Empirical Data. *SIAM reviews*, june 2007.
13. B. Decocq. *Résolution d'un problème de placement de processus sur calculateurs et de fichiers sur disques EDF*. PhD thesis, CNAM/EDF, 1996.

Appendix

As usual in Mixed Integer Programming, several models (sets of inequalities) describing one problem are not equivalent to each other. Some of them are

far more efficient. By describing the four models in the appendix, we want to permit the interested reader to compare efficiencies of different MIP models solving problem ($\mathcal{P}$). The constraints and variables (1) to (7) on page 4 are common to the four models. For the second model, constraints and objective function (6) to (10) are replace by (11) to (13). It introduces two slack variables $u_j, l_j \geq 0$, and aims at minimizing sum of those two slack variables:

$$\sum_{j \in J} x_{i,j} \geq 1 \ \forall i \in I \tag{11}$$

$$S_j - u_j + l_j = \frac{1}{\|I\|} \sum_{i \in I} (N_i) \ \forall j \in J \tag{12}$$

$$\text{minimize} \sum_{j \in J} u_j + l_j \tag{13}$$

For the third model, constraints and objective function (6) to (10) are replaced by (14) to (17). It introduces four slack variables $S_{\min}, S_{\max}, T_{\min}, T_{\max}$:

$$\sum_{j \in J} x_{i,j} = 1 \ \forall i \in I \tag{14}$$

$$S_{\min} \leq S_j \leq S_{\max} \ \forall j \in J \tag{15}$$

$$T_{\min} \leq T_j \leq T_{\max} \ \forall j \in J \tag{16}$$

$$\text{minimize } S_{\max} - S_{\min} + k * (T_{\max} - T_{\min}) \tag{17}$$

For the fourth model, constraints and objective function (6) to (10) are replaced by (18) to (20). It introduces only one slack variable m and tries to maximize it:

$$\sum_{j \in J} x_{i,j} \leq 1 \quad \forall i \in I \tag{18}$$

$$S_{j_1} + k * T_{j_2} \geq m \ \forall j_1, j_2 \in J \tag{19}$$

$$\text{maximize } m \tag{20}$$

Efficiency of the four models is shown in Fig. 5 on a 200 seconds example. The most efficient model, say model F2, is the one presented in the present article.

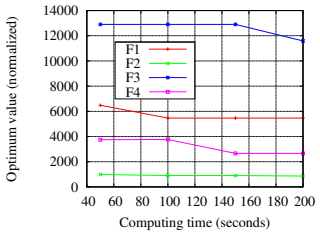


Fig. 5. Convergence of the 4 models.