Storing and Replication in Topic-Based Publish/Subscribe Networks

Vasilis Sourlas*†, Paris Flegkas*†, Georgios S. Paschos*† and Leandros Tassiulas*†
*Department of Computer & Communication Engineering
University of Thessaly, Greece, †CERTH-ITI
Email: vsourlas, gpasxos, pflegkas, leandros @inf.uth.gr

***Abstract*—In current publish/subscribe networks messages are not stored and only active subscribers receive published messages. However, in a dynamic scenario a user may be interested in content published before the subscription time. In this paper, we introduce a mechanism that enables storing in such networks, while maintaining the main principle of loose-coupled and asynchronous communication. Furthermore, we propose a new storage placement and replication algorithm which differentiates classes of content and minimize the clients response latency. The performance of our proposed placement algorithm and the proposed storing mechanism is evaluated via simulations and insights are given for future work.**

I. Introduction

Applications that exploit a publish/subscribe (pub/sub) communication paradigm are organized as a collection of autonomous components, the clients, which interact by publishing events and by subscribing to the classes of events they are interested in. A component of the architecture, the event dispatcher (or rendezvous point or event broker or simply broker), is responsible for collecting subscriptions and forwarding events to subscribers. In pub/sub networks, the selection of a message is determined entirely by the client, which uses expressions (filters) that allow sophisticated matching on the event content.

In a pub/sub network, any message is guaranteed to reach all interested active clients whose subscriptions are known to the network at publish time. However, in a dynamic distributed environment, clients join and leave the network during time, and it is possible that a client joins the network after the publishing of an interesting message. In current pub/sub systems, it is not possible for a new subscriber to retrieve previously published messages that match his/her subscription. Therefore, enabling the retrieval of previously published content by means of storing is one of the most challenging problems in pub/sub networks.

Data storage servers or simply "storages" replicate the whole content of a given server, unlike caches where misses could occur. When a client is interested in the content of that server, his/her request is redirected to one of the existing storages (i.e. the closest one). Since storages serve only a portion of the total requests and are placed closer to the client, clients are served faster. A client's request is redirected to a storage only if that storage is a replica of the targeted server otherwise the request is served by the server itself.

In this paper, we describe and evaluate through simulations our design of a storing technique on networks that use the topic-based pub/sub communication paradigm. Particularly we enhance the pub/sub paradigm with an advertisement and a request/response mechanism so that storages can advertise the class of the content (*topic*) that they have stored and clients can retrieve that stored content. We also propose a new algorithm for the selection of M storage points among the N brokers ($M < N$) based a) on the locality of the interest for each topic, b) the targeted replication degree of each topic (as replication degree we name the number of replicas $k \leq M$ of the topic among the storages) and c) the capacity limitations CL for each storage. The objective function of our scheme is to minimize client's response latency subject to installing the minimum number of storages in the network.

The rest of the paper is organized as follows. In section II, a brief introduction of storing in pub/sub architectures is given, followed by a brief description of the storage placement problem. In section III, we shortly describe the pub/sub architecture and present the proposed advertisement and request/response mechanism. The new algorithm for the selection of the storage location and the replication of the content is presented in section IV while section V is devoted to performance evaluation via simulations. Finally, we conclude the paper and give insights for future work in section VI.

II. Related work

Despite the fact that there are several research efforts concerned with the development of an event notification service, like IBM's Gryphon [1], Siena [2] and REDS [3], storing data through replication has not received attention in the literature. Only in [4], the authors propose a historic data retrieval pub/sub system where databases are connected to various brokers, each associated with a topic to store. The database holding the relevant information republishes historic events on receipt of a subscription query with a time-based parameter. In the same work, every database advertises the class to the network so that subscribers interested in past information belonging to that class can request it from the specific database. In [4] every message is stored only once and no placement strategies have been examined while there is no description of the mechanism for the retrieval of the stored data.

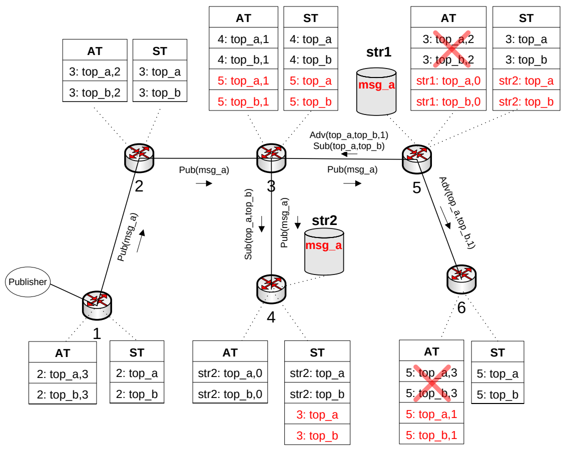


Fig. 1. Advertising and Storing of information (in red are the new entries of STs and ATs created by the installation of the "str1"). Publisher at broker 1 publishes a message msg_a that matches top_a and is stored at "str1" and "str2".

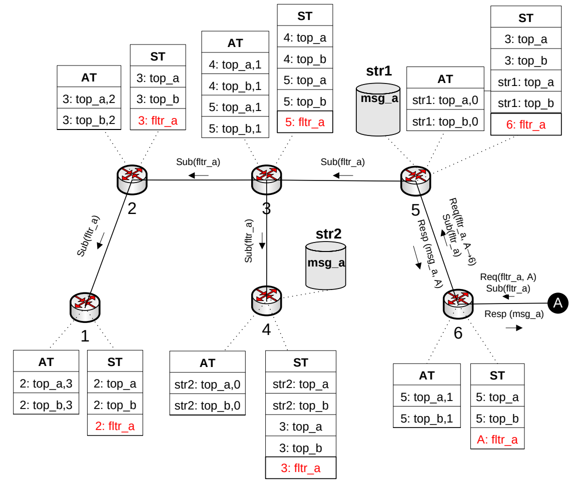


Fig. 2. Retrieval of stored information using the request/response mechanism (in red are the new entries of STs created by the subscription of client A).

On the other hand the placement problem, especially in the context of Content Delivery Networks and Web Proxies, is a thoroughly investigated problem. Particularly in [5]-[6] authors approached the placement problem with the assumption that the underlying network topologies are trees. The placement problem is in fact an NP-hard problem when striving for optimality, but there is a number of studies [7]-[12] where an approximate solution is pursued. Their work is also known as network location or partitioning and involves the optimal placement of k service facilities in a network of N nodes targeting the minimization of a given objective function. If the objective is to minimize the overall traffic incurred by messages sent among the service facilities themselves and the messages sent between clients and the service facilities then the location problem is reduced to the k-*median* problem, while if the objective is to minimize the maximum response time the problem is reduced to the k-*center* problem.

Moreover, in [13] authors investigated the QoS-aware replica placement problems for responsiveness QoS requirements. They considered replica aware services (servers are aware of the replicas and can optimize request routing) and replica-blind services. They show that QoS-aware placement problem for replica-aware services is NP-complete and they proposed several heuristic algorithms for fast computation of good solutions. Also they proposed efficient algorithms to compute optimal locations of replicas in the case of replica-blind services. Finally, in [14] authors introduce a framework for evaluating placement algorithms. Firstly, they classify and qualitatively compare placement algorithms using a generic set of primitives that capture several objective functions and near optimal solutions, while secondly provide estimates for their decision time using an analytic model. The model takes into account not only computational complexity and message numbers but also memory constraints (disk accesses and message sizes) to produce good estimates.

III. Enabling storing in pub/sub networks

We consider a pub/sub network which uses the subscription forwarding routing strategy [2]. The routing paths for the published messages are set by the subscriptions, which are propagated throughout the network so as to form a tree that connects the subscribers to all the brokers in the network. Publishers join the network when they have something to publish, therefore in our approach the entity of the server does not exist.

In a pub/sub network when a client issues a subscription, a message containing the subscription filter is sent to the broker the client is attached to. The filter is inserted in a Subscription Table (ST), together with the identifier of the subscriber. Then, the subscription is propagated by the broker, which now behaves as a subscriber with respect to the rest of the dispatching network, to all of its neighboring brokers on the network. In turn, the neighbors record the subscription and re-propagate it towards all further neighboring brokers, except for the one that sent it. Finally, each broker in the network has a ST, in which for every neighboring broker there is an associated set of filters containing the subscriptions sent by them.

A. Advertisement and Request/Response mechanism

By installing storages and introducing an advertisement and a request/response mechanism, we aim to provide a pub/sub system with the ability to store and retrieve information published in the past and make it available for future clients. We will present through the example of figures 1 and 2 the new mechanism.

In order to retrieve old information, we add to the system three additional types of messages, `Advertise()`, `Request()` and `Response()`. When a new storage "*str1*" is attached at broker 5 (fig. 1) issues a `Subscribe()` message with the topics (class of events) that is willing to store (top_a and top_b). In that way, it acts as a client to future publications matching the subscribed topics and each time a publication occurs (publisher attached to broker 1 publishes message msg_a matching top_a) stores the message (the message is also stored to *str2*). The "str1" also issues an `Advertise()` message which contains the topics that stores and the distance in hops from the storage. Advertisements are treated similarly to subscription messages so as to form a tree that connects the "str1" to all the brokers in the network. Advertisements are inserted in the `Advertisement Table` (`AT`) a new feature similar to ST that we also added to our approach. Coverage also occurs with advertisements, like subscriptions, but in a slightly different way. Particularly, when a broker receives an advertisement checks in the distance field and if the distance is equal to another entry (for the same topics) adds the advertisement to the AT and stops forwarding the advertisement (broker 3 in fig. 1). Keeping more than one entries for the same topic in an AT, enables load balancing capabilities to requests passing from that particular broker. On the other hand, when a broker receives an advertisement for a storage which is closer compared to the other storages already in the AT, adds the advertisement to the AT, removes the previous entries and forwards it further (brokers 5 and 6 in fig. 1). Finally, when a broker receives an advertisement for a storage which is further compared to the other storages already in the AT simply stops the forwarding of the advertisement without changing the AT.

When a client node (client A in fig. 2), interested in old (and probably new) content, appears in the network, apart from subscribing (for future publications) also makes a request by sending a `Request()` message containing the interested filter ($fltr\_a$). The filter contains the topic that the client is interested in. Filters are identical to topics but they can contain more attributes to enable more sophisticated match than simply using the topic. We use source routing for the forwarding of the `Request()` (the path is being built hop by hop and is included in the `Request()` header). Broker 6 upon receiving the `Request()` message checks in its Advertisement Table (AT) for entries matching the requested topic (top_a in this case). The broker forwards the `Request()` message to the broker who had advertised the matching topic and is closer to the client (in this example broker 5). Finally, "str1" receives the `Request()` message, matches its stored content with the whole filter (not just the topic) and initiates a `Response()` message for each match (messages msg_a in fig. 2).

A `Response()` message carries a stored message as well as the sequence of nodes carried by the initiating `Request()` message (source routing). When a broker receives a `Response()` message, pops off its identifier from that sequence and forwards it to the first broker of the remaining sequence. In the end, client A will receive every stored message matching its filter.

IV. Placement/Replication Strategy

In [7], authors developed several placement algorithms that use workload information, such as latency (distance from the storage points) and request rates, to make the placement decision. Their main conclusion is that the so called "greedy" algorithm that places storages based upon both a distance metric and request load, performs the best and very close to the optimal solution.

A. Greedy algorithm

In this section, we briefly describe the greedy algorithm, which is the base of our placement algorithm, assuming that there exists only one class of content in our system, or equivalently there is no distinction in the content. We let ρ_i be the traffic (in reqs/sec) from clients attached to node i. We also let P_{ij} be the percentage of the overall traffic accessing the target server j (traditional placement algorithms replicate a specific server) that passes through node i. Also we let the propagation delay (hops) from node i to the target server j as D_{ij}. If a storage is placed at node i we define the Gain to be $G_{ij} = P_{ij} * D_{ij}$. This means that the P_{ij} percentage of the traffic would not need to traverse the distance from node i to server j.

The greedy algorithm chooses one storage at a time (we need k storages out of N nodes). In the first round evaluates each of the N nodes to determine its suitability to become a storage (replication point of server j). It computes the Gain associated with each node and selects the one that maximizes the Gain. In the second round, searches for a second storage which, in conjunction with the storage already picked, yields the highest Gain. Each request uses a single storage, we assume in other words full replication of the content among the selected storages. The greedy algorithm iterates until k storages have been chosen for the specific server j.

In our work, we have no knowledge of the location of the server, or differently there is no such a server. Publishers join the network publish their content and disappear. So we repeat the above procedure N times assuming each time that the targeted server j is a different node (broker) of the network. We get in that way N vectors of k possible storages. Precisely each vector has N elements with k ones in the index of the selected storages and $N - k$ zeros in every other place (for example vector $[0\ 0\ 1\ 0\ 1]$ means that of the 5 nodes of the network the selected $k = 2$ possible storages are nodes 3 and 5). In two different vectors there might be subsets of possible storages present in both vectors. Finally, we select as our storages those k nodes that appeared more times in the per element summation of the N vectors and install at each one a storage following the mechanism described in III-A.

B. Placement algorithm for pub/sub networks

In this section, we describe a modified version of the greedy algorithm described above for the case where in our network exist T different classes of content (topics). If we assume that each storage has capacity limitations equal to CL different topics and each topic should be replicated k times, then our algorithm should select M storages and assigns each topic at exactly k different storages. The storage limitations usually refers to TBytes but for simplicity we assume here that messages are published with the same rate for each topic and messages are of the same size. In other words at every time instance in the network exist the same number of messages for each topic. The CL parameter is a limitation introduced by the storage providers and refers to the maximum storing capability of each storage in the network. On the other hand the k parameter (replication degree of each topic) is a limitation introduced by the content providers of the network and refers to the maximum number of replicas that the content provider is willing to pay for. Finally, the M parameter refers to the minimum number of storages that a storage provider should install in the network to serve the storage demands. Our algorithm is composed by the following steps:

1) For each topic we execute the greedy algorithm presented in IV-A and we get T vector of possible storages.
2) Each vector is weighted by $w_t = \sum_{i=1}^{N} \rho_i^t / \sum_{i=1}^{N} \sum_{t=1}^{T} \rho_i^t$. w_t shows the significance regarding the traffic of each topic in the network.
3) We select as our storages those M nodes that appeared more times in the per element weighted summation of the T vectors. We call that vector as *storage brokers vector SBV*.
4) For each topic t starting from the most significant (based on the weight) assign k storages following the below procedure:
   - For each entrance in the vector of the topic t calculated in step 1 assign a storage if that entrance also appears in the *SBV* calculated in step 3 and only if in that storage has been assigned less than CL topics until we get k or less storages (replication of topic t).
   - If the number of the assigned storages for each topic t is less than k then assign t, until we get k storages, to the storage brokers in *SBV* that still has less than CL assignments.

Below is an example of the placement algorithm for the pub/sub network for the network of figure 3 assuming $k = 2$, $CL = 2$ and $T = 3$, meaning that we should select $M = 3$ storage points out of the $N = 6$ brokers of the network.

- Step 1 produces vectors $[0\ 3\ 5\ 0\ 2\ 2]$, $[0\ 2\ 5\ 0\ 5\ 0]$ and $[0\ 2\ 5\ 0\ 5\ 0]$ for the three topics accordingly (the $[0\ 3\ 5\ 0\ 2\ 2]$ means that out of the $N = 6$ executions of the greedy algorithm node 2 appeared 3 times, node 3 appeared 5 times and so on).
- The weights regarding step 2 are $w_a = 17/50 = 0.34$, $w_b = 27/50 = 0.54$ and $w_c = 6/50 = 0.12$. So the vectors from step 1 are transformed to $[0\ 1.02\ 1.7\ 0\ 0.68\ 0.68]$, $[0\ 1.08\ 2.7\ 0\ 2.7\ 0]$ and $[0\ 0.24\ 0.6\ 0\ 0.6\ 0]$ accordingly.
- The per element summation of those three vectors into a single vector for step 3 gives $[0\ 2.34\ 5\ 0\ 3.98\ 0.68]$ meaning that the final $M = 3$ storages in *SBV* are nodes 3, 5 and 2.
- For each topic starting from topic b then topic a and finally topic c (based on their weights) we assign them to $k = 2$ storages. Topic b is assigned to nodes 3 and 5 which were the nodes for topic b appeared more times by step 1. Topic a is also assigned to nodes that were produced by step 1, nodes 2 and 3, while topic c is assigned to nodes 2 and 5. Node 5 was among the most popular selections produced by step 1 while node 2 was the only storage in *SBV* with less than $CL = 2$ assignments.

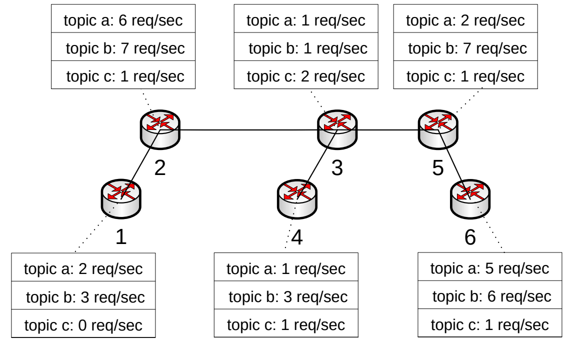


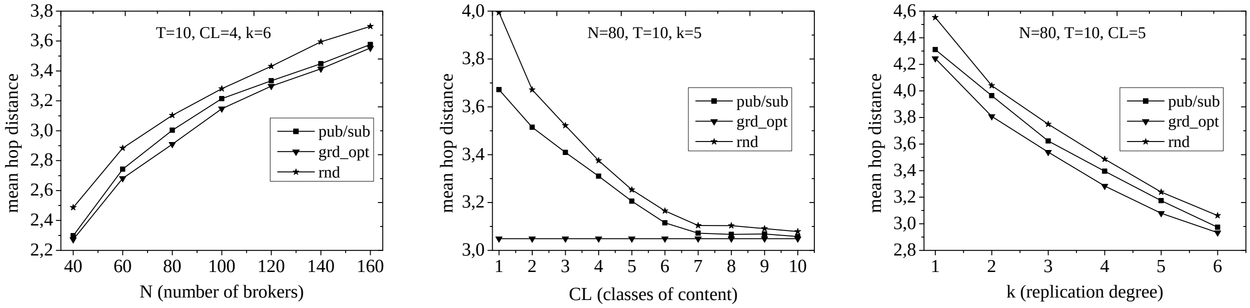
Fig. 3. Topology and workload information (requests/second) per each class of content (3 topics) together with $k = 2$, $CL = 2$ and $M = 3$ form the inputs of the placement algorithm for the pub/sub network.

Step 4 of our algorithm is also known as the *Generalized Assignment Problem* which even in its simplest form is reduced to the NP-complete multiple knapsack problem. In this paper for the solution of the assignment problem we used the heuristic approach described above, while more approaches could be found in literature [15]-[16].

V. Performance evaluation

In this section, we evaluate the proposed mechanism using a discrete event simulator. N brokers are organized in a tree topology (common topology in overlay pub/sub networks) and clients dynamically request on each broker i for stored content with rate ρ_i^t different for each topic t. We assume that in our network exist T topics that should be replicated exactly k times and each storage has a capacity limitation of CL different topics. For the purpose of this paper, we assume that there are no limits in the workload (in requests/second) that each storage can serve. New publications occur to the network with rate λ_{msg} equal for every class of content, while stored messages are removed from the storages with rate μ_{msg}. The removal of a message corresponds to the expiration of the life time of that message which is typical in every data storage scheme.

Having selected the M storages and assigned to them the T topics using our placement algorithm for pub/sub networks ("*pub/sub*") we let the system operate under the dynamic client environment. We compare it firstly to the case where each topic is assigned to the k storages produced by the first step of the placement algorithm ("*grd_opt*") described in IV-B disregarding of the capacity limitations and the total number M of used storages, and secondly to the case where there is no differentiation among topics during the selection of the M storages and the final assignment of the topics to k storages is random ("*rnd*"). The metric we are interested in is the *mean hop distance* which corresponds to the mean number of hops between a responding storage and the client making



(a) (b) (c)

Fig. 4. Performance of the proposed placement algorithm compared to the "grd_opt" and the "rnd".

the request. This metric is indicative of the response latency as a function of hops in the network.

We set three experiments, one varying the number of brokers in the network N, one varying the capacity limitations CL of each storage and one varying the replication degree of the content in the network k. We also assume that in our networks exist $T = 10$ different topics, clients request rate per topic vary between 0-1 requests/second for each broker, new messages are published in the network with rate 1 message/second per topic, while finally the lifetime of each message is set to $1/\mu_{msg} = 1000$ seconds.

Subfigure "a" of figure 4 shows the mean hop distance depending on the number of brokers in the network. The proposed "pub/sub" algorithm behaves better than the "rnd" algorithm and close to the "grd_opt". Subfigure "b" shows the mean hop distance depending on the storage limitation of each storage. The proposed "pub/sub" algorithm behaves better than the "rnd" algorithm and close to the "grd_opt" as previously, while when the CL is close to the number of topics (T) the three algorithms perform the same regarding the mean hop distance. This happens since now the algorithms select almost the same storage brokers. Finally, subfigure "c" shows the mean hop distance depending on the replication degree of the content. It is obvious that the mean hop distance is decreasing as the number of replicas increases since now requests reach closer storages.

VI. Conclusion And Future Work

In this paper, we put forward a new mechanism for storing in topic-based pub/sub networks. The proposed concept equips the pub/sub with the ability to store and retrieve stored information. Moreover, we presented a new placement and replication algorithm that differentiates classes of content. Evaluation via simulations presents the performance of the system regarding the clients response latency. This work can be extended in many ways from optimizing different objective functions and dynamic assignment of topics among the storages to provide differentiation among the classes of content.

References

[1] Aguilera M. K., Strom R. E., Sturman D. C., Astley M. and Chandra T. D., "Matching events in a content-based subscription system," 18th ACM Symposium on Principles of Distributed Computing (PODC '99) Atlanta, GA, May 4-6, pp. 53–61, 1999.

[2] Carzaniga A., Rosenblum D. and Wolf A., "Design and evaluation of a wide-area event notification service," ACM Transaction On Computer Systems, vol. 19, pp. 332–383, 2001.

[3] Cugola G. and Picco G., "REDS, A Reconfigurable Dispatching System," 6th International workshop on Software Engineering and Middleware, pp. 9–16, Oregon, 2006.

[4] Li G., Cheung A., Hou S., Hu S., Muthusamy V., Sherafat R., Wun A., Jacobsen H., and Manovski S., "Historic data access in publish/subscribe," Proceedings of the 2007 inaugural international Conference on Distributed Event-Based Systems (DEBS 2007), pp. 80–84, Toronto, Canada, 2007.

[5] B. Li, M. J. Golin, G. F. Ialiano and X. Deng, "Dynamic and Distributed Web Caching in Active Networks," Asia Pacific Web Conference (AP-Web'98), Beijing, China, Sept. 1998.

[6] B. Li, M. J. Golin, G. F. Ialiano and X. Deng, "On the Optimal Placement of Web Proxies in the Internet," In Proc. of INFOCOM99, March 1999.

[7] L. Qiu, V.N. Padmanabhan and G. Voelker, "On the placement of web server replicas," In Proc. of IEEE INFOCOM'01, Anchorage, AK, USA, April 2001.

[8] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala and V. Pandit, "Local search heuristics for k-median and facility location problems," In Proc. of 33rd ACM Symposium on Theory of Computing, 2001.

[9] M. Charikar and S. Guha, "Improved combinatorial algorithms for facility location and k-median problems," In Proc. of the 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 378-388, October 1999.

[10] M. Charikar, S. Khuller, D. Mount, and G. Narasimhan, "Facility location with outliers," In 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington DC, January 2001.

[11] D.B. Shmoys, E. Tardos and K.I. Aardal, "Approximation algorithms for facility location problems," In Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pp. 265-274, 1997.

[12] E. Cronin, S. Jamin, C. Jin, T. Kurc, D. Raz and Y. Shavitt, "Constrained mirror placement on the internet," IEEE Journal on Selected Areas in Communications, 36(2), September 2002.

[13] X. Tang and J. Xu, "QoS Aware Replica Placement for Content Distribution," IEEE Trans. Parallel and Distributed Systems, pp. 921-932, vol. 16, no. 10, Oct. 2005.

[14] M. Karlsson, Ch. Karamanolis and M. Mahalingam, "A Framework for Evaluating Replica Placement Algorithms", http://www.hpl.hp.com/techreports/2002/HPL-2002-21, 2002.

[15] R. Cohen, L. Katzir and D. Raz, "An Efficient Approximation for the Generalized Assignment Problem," Information Processing Letters, Vol. 100, Issue 4, pp. 162-166, November 2006.

[16] L. Fleischer, M. X. Goemans, V. S. Mirrokni and M. Sviridenko, "Tight Approximation Algorithms for Maximum General Assignment Problems," SODA 2006, pp. 611-620.