Performance Evaluation of Content Distribution Network Architectures through Agent-Based Modeling and Simulation

Giancarlo Fortino*, Alfredo Garro, Wilma Russo, Marino Vaccaro
Dipartimento di Elettronica, Informatica e Sistemistica (DEIS)
Università della Calabria
Via P. Bucci, cubo 41C, 87036, Rende (CS), Italy
{g.fortino, garro, w.russo}@unical.it

***Abstract*—Agent-Based Modeling and Simulation (ABMS) has emerged as a new and powerful technology for the analysis of natural and artificial complex systems. In this paper ABMS is exploited for the modeling and performance evaluation of Conventional, Clustered and Cooperative Content Distribution Network (CDN) architectures. Clustered and Cooperative architectures differ from Conventional architectures as surrogate servers can loosely (in the Cooperative architectures) or tightly (in the Clustered architectures) cooperate to provide the requested contents to users. The results obtained from the simulation phase show that the Clustered architectures allow for significant improvements of the main CDN performance indices (average user perceived latency, cache hit ratio, and CDN utility) with respect to Conventional and Cooperative architectures.**

Keywords-component; Content Delivery Networks, Surrogate Clustering, Agent-based Modeling and Simulation, Performance Evaluation.

I. INTRODUCTION

The analysis and design of modern distributed systems require powerful and flexible methods, tools and techniques, which are also based on bottom-up approaches and incorporate the use of simulation to support the typical phases of a software engineering process [10]. In this context, a suitable support can be represented by the possibility of obtaining agent-based models of systems, i.e. system models which are built through a bottom-up approach in terms of proactive and reactive autonomous entities that dynamically interact and cooperate with each other [9]. An agent-based model of a system can then be simulated, so to observe emergent macro-level phenomena hard to catch with other analysis techniques, and can be used to validate and evaluate different design choices at architectural and behavioral levels [11]. Moreover, the agent-based model of the system exploited during the design phase can be used as a starting point for an agent-based system implementation [3].

In this paper, the Agent-Based Modeling and Simulation (ABMS) approach is exploited for modeling and evaluating through simulation different Content Distribution Networks (CDNs) architectures that represent effective solution for improving the performance of content delivery by means of coordinated content replication [2]. In particular, five distributed architectures (*Conventional*, *Cooperative*, *Master/Slave*, *Multicast-based*, *Peer-to-Peer*), which are enhanced with respect to those presented in [6], have been modeled and extensively evaluated.

The *Conventional* architecture is based on the basic schema of CDN, *Master/Slave*, *Multicast-based*, and *Peer-to-Peer* architectures (or *Clustered* architectures) relies on a new schema based on clustering [6], whereas the *Cooperative* architecture depends on non coordinated cooperation among neighbor surrogate servers. In particular, the *Clustered* architectures differ from the *Conventional* architecture as surrogate servers, which cache the content originally produced and stored in *origin* servers so to improve performances in delivering contents to users, are grouped into clusters of neighbor surrogates which can cooperate to provide requested contents. Specifically, a surrogate that is not able to provide the requested content does not directly ask the *origin* server for it as in the *Conventional* architecture; the surrogate first checks for a surrogate of the same cluster having the content so as to forward the unfulfilled user request to it. Differently, in the *Cooperative* architecture the surrogate asks its neighbors for the requested content and, upon content attainment, replies to the requesting user. Finally, both in *Clustered* and *Cooperative* architectures, if the surrogate is not able to find the content in its cluster or among its neighbors, it contacts the *origin* server as in the *Conventional* architecture.

To analyze the performances of the five considered architectures with respect to the most important CDN performance indices (average perceived user latency [13], cache hit ratio [13], and CDN utility [18]), for each architecture, an agent-based model has been defined and simulated by using the ELDAMeth methodology [5]. In particular, the modeling phase is driven by a Statecharts-based modeling language and supported by a CASE tool (ELDATool) which automatically translates visual specifications into platform-independent code; the simulation phase is based on an agent-oriented and event-driven simulation framework (ELDASim) which executes the code produced in the modeling phase in the context of purposely set distributed scenarios.

The results obtained from the performance evaluation phase show that the *Clustered* architectures can provide higher

*corresponding author

performances, in terms of the considered performance indices, than those of *Conventional* and *Cooperative* architectures.

The rest of the paper is organized as follows. Section 2 introduces the considered distributed architectures for content delivery. Section 3 is devoted to the agent-based simulation of the presented architectures and, in particular, to the analysis of the performance evaluation results. Finally conclusions are drawn and directions of future research briefly elucidated.

II. CONVENTIONAL, COOPERATIVE AND CLUSTERED CDN ARCHITECTURES

In *Conventional* CDN architectures [14], which are based on stand-alone *Surrogates*, when a *Client* issues a content request, a *Redirection System* selects the most appropriate Surrogate (typically the *closest* to the Client) to which the request is routed. The selected Surrogate serves the request if the requested content is available; otherwise it asks the *Origin* Server for the content and, once retrieved, sends it to the requesting Client. Although this *Conventional* architecture (hereafter named *SimpleArch*) is easy to develop and maintain, it suffers of two main drawbacks: limited dimension of the cache of the surrogates and high response time when not cached content must be fetched from the Origin Server.

To deal with these drawbacks, Surrogates can be grouped into loosely-coupled and tightly-coupled clusters according to their proximity (e.g. neighboring Surrogates belong to the same cluster) [4, 8, 12, 17]. Surrogates of the same cluster (hereafter referred as peers) cooperate with each other to provide contents when they cannot directly serve content request. This would result in a higher hit rate, as the available content is at most the total content of all peers and not just the content of a single Surrogate, and in a shorter response time, as distance between peers is much shorter than the distance between Surrogates and their related Origin server. Indeed, a request is forwarded to the Origin Server only if none of the peers can provide the requested content.

In the following, four different distributed architectures for surrogate clustering are presented. In particular, the three tightly-coupled *Clustered* architectures, Master/Slave (*M/SArch*), Multicast-based (*MCArch*), and Peer-to-Peer (*P2PArch*), are described in Sections II.A-C, respectively; in Section II.D the loosely-coupled *Cooperative* architecture (*CoopArch*) is defined.

A. Master/Slave Architecture

In the *M/SArch* architecture [6], a master/slave approach is exploited which is based on a master peer to manage the cluster *Content Location Hash Table* (CLHT) whereas the other peers only manage a CLHT of their own content. When a request arrives, the selected surrogate looks up its CLHT and then, if the content is not found, forwards the request to the master peer that, in turn, forwards it either to the peer (which could also be the master itself) with that content or to the origin server. It is worth noting that every time a peer chooses to evict a content, it notifies the master that consequently updates the global CLHT. In this way consistency of the cluster is guaranteed by the master even though it could become a bottleneck.

Four variants of *M/SArch* (*M/SArch* _1, *M/SArch*_2, and *M/SArch*_3, *M/SArch*_4) have been defined on the basis of different schemas related to *content found*/*content not found in the cluster* scenarios. Specifically, two schemas for the *content found in the cluster* scenario are considered: in the first, which is exploited in *M/SArch*_1 and *M/SArch*_2, the master peer forwards the request directly to the surrogate which has the content; in the second, which is exploited in *M/SArch*_3 and *M/SArch*_4, the master peer notifies the address of the surrogate which has the content to the selected surrogate which, in turn, forwards the request to it. Two schemas for the *content not found in the cluster* scenario are also introduced; in the first, which is exploited in *M/SArch*_1 and *M/SArch*_3, the master peer replies to the selected surrogate which, in turns, downloads the content from the origin to serve the client; in the second, which is exploited in *M/SArch*_2 and *M/SArch*_4, the master peer contacts the origin which sends the missing content to the selected surrogate.

It is worth noting that the four variants of *M/SArch* consider the architectures that averagely involve the lowest number of exchanged messages (*M/SArch*_2), the highest number of exchanged messages (*M/SArch*_3), and a number of exchanged messages between the highest and lowest ones (*M/SArch*_1, *M/SArch*_4).

B. Multicast-based Architecture

In the *MCArch* architecture [6], each peer surrogate manages a CLHT in which stores the content location information of all peer surrogates. A missing content in the selected peer is handled as follows: if the CLHT has an entry for that content, the request is forwarded to the peer that has the requested content and will then serve the client request; otherwise, the request is forwarded to the origin server and then handled as in the *Conventional* architecture. Every update of the CLHT is multicast from the peer that updated its content to all the others that consequently update their CLHT without an ACID (Atomicity, Consistency, Isolation, e Durability) coordination mechanism. This implies that the consistency of the CLHT is not guaranteed and then a peer could forward a request to another peer that may not have the requested content; moreover, duplicated copies of the same content could be present in a cluster.

C. Peer-to-peer Architecture

In the *P2PArch* architecture [6], each peer has an SLT (Surrogate Location Table) which contains the location information of all the peers and their respective contents. In particular, for each peer surrogate an SLT has an entry formalized by the pair <SId, CZ>, where CZ (Content Zone) is the space of the identifiers of the contents potentially stored in the peer identified by SId. According to the peer-to-peer model, a content request issued by a client is served by the selected surrogate as follows: (i) if the CId of the requested content belongs to its CZ, the content is looked up in the CLHT and then, if the content is present, it is sent to the client; otherwise, the content is retrieved from the origin, sent to the client and finally stored; (ii) if the CId of the requested content does not belong to its CZ, the request is forwarded to the peer responsible for Cid and then, if the requested content is present,

it sent it to the requesting client; otherwise, the content is fetched from the origin before sending it to the requesting client.

As in the *M/SArch* architecture, the *P2PArch* provides consistency of the content in the cluster. Moreover it overcomes the main drawback of *M/SArch* and *MCArch* as a peer does not need to maintain content information belonging to the other peer surrogates.

D. Cooperative Architecture

In the *CoopArch* architecture, which is based on the cooperative architecture proposed in [19], each peer has a given number of *Neighbor* surrogates and manages a CLHT in which stores the content location information of its *Neighbors*. When a peer is not able to provide the requested content but the CLHT has an entry for it, the content is asked to the associated Neighbor otherwise it is asked to the Origin Server. Upon the reception, the content is stored in the peer cache (removing another content if necessary) and sent to the user; as a consequence a content can be replicated on different surrogates. The CLHT of the peer is then updated and this information is sent to all the *Neighbor* surrogates of the peer so that they can consequently update their CLHT.

III. Performance Evaluation Through Agent-Based Modeling and Simulation

The distributed architectures presented in Section II have been modeled as Multi-Agent Systems (MASs) by exploiting the ELDAMeth methodology and related tools that allow a visual and Statecharts-based modeling of the agent behaviors and the semi-automatic translation of the visual specifications into code which can be executed by an agent-oriented and event-driven simulation framework [5]. Specifically, the aim of the simulation was to analyze the efficiency of the proposed distributed architectures in terms of the following three indices which mainly characterize the performances of content delivery networks:

- Average user perceived latency (*AUPL*) [13], which is defined as the average time ranging from the transmission of the content request by a client (t_{req}) and the reception of the content (t_{serv}):

$$AUPL = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{n_{req_i}}\sum_{j=1}^{n_{req_i}}\left(t_{i,req_j} - t_{i,serv_j}\right)\right) \qquad (1)$$

  where N represents the number of surrogates and n_{req_i} the number of requests issued to the surrogate i.

- Cache hit ratio (*CHR*) [13], which is defined as the percentage of content requests successfully served by the CDN without fetching content from the Origin Server:

$$\mathrm{CHR} = \frac{\sum_{i=1}^{N} locserv_i}{\sum_{i=1}^{N} req_i} \qquad (2)$$

  where N represents the number of surrogates, *req* the amount of bytes requested to a surrogate and *locserv* the amount of bytes served by a surrogate without fetching content from the origin server.

- Utility (*UT*) [18], which is defined in terms of the byte amount that the surrogates of the CDN send to the requesting clients and receive from the origin server and/or other surrogates. Specifically, in the case of *Non Clustered* (*NC*) architectures, i.e. *SimpleArch* and *CoopArch*, UT is evaluated as follows:

$$UT_{NC} = \frac{\sum_{i=1}^{N} UT_i}{N} \qquad (3)$$

  where N represents the number of surrogates in the CDN and $UT_i = \frac{2}{\pi}\times\arctan\left(\xi_i\right)$ is the utility of the surrogate i which depends on the ratio $\xi_i = \frac{up_i}{dw_i}$ of the bytes sent to clients (up_i, upload to clients) and received (dw_i, download from the origin server and/or other surrogates) by the surrogate i.
  Conversely, in a *Clustered* (C) CDN (*M/SArch*, *MCArch* and *P2PArch*), as surrogates can download contents only by the Origin Server and the cluster can be therefore seen as a unique surrogate with a cache equals to the sum of the caches of the peer surrogates (see Section II), UT is calculated as:

$$UT_C = \frac{2}{\pi}\times\arctan\left(\Xi\right) \qquad (4)$$

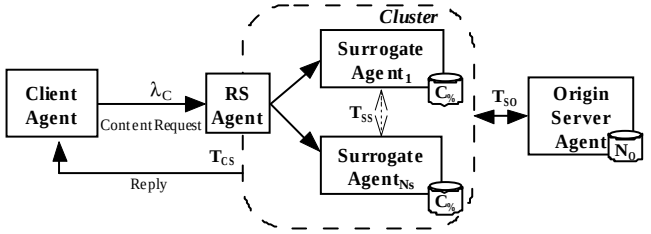
  where $\Xi = \frac{up_c}{dw_c}$ is the ratio of the bytes sent to clients (up_c) and received (dw_c) by the whole cluster of N surrogates; $up_c = \sum_{i=1}^{N} up_i$ and $dw_c = \sum_{i=1}^{N} dw_i$ are respectively obtained by summing up the amount of bytes uploaded from and download by each surrogates of the cluster.

In the next subsections, after briefly discussing the agent-based modeling of the CDN architectures (Section III.A), simulation parameters are introduced (Section III.B) and the analysis of the obtained simulation results (Section III.C) is presented with reference to different and significant simulation settings.

A. Agent-based modeling

In order to evaluate the defined performance indices (AUPL., CHR and UT), agent-based models of the *SimpleArch*, *CoopArch*, *M/SArch*, *MCArch* and *P2PArch* architectures (see Section II), have been defined according to the reference schema reported in Figure 1 which shows four type of *Agents* representing the basic components of every CDN architecture: *Client Agent*, *Redirection System Agent*, *Surrogate Agent* and *Origin Server Agent*. Specifically, when a *Client Agent* request

is generated, it is forwarded to a *Surrogate Agent* randomly selected by the *Redirection System (RS) Agent*; then, such request is handled according to the different considered architectures. The reference schema was defined to evaluate a single cluster scenario so that the considered *Surrogate Agents* adhere to either a *Clustered* or *Non Clustered* (i.e. *Conventional* or *Cooperative*) architecture. A single cluster scenario allows a straightforward comparison of the considered architectures and a generalization of the obtained results to a CDN composed of different clusters.



LEGENDA
N_O: number of objects contained in the origin server
$C_{\%}$: percentage of objects that are stored in a surrogate
T_{CS}: average latency time between clients and surrogates
T_{SS}: average latency time among surrogates
T_{SO}: average latency time between surrogates and origin server
N_S: number of surrogates
λ_C: average rate of client requests

Figure 1. The reference CDN agent-based model.

According to the ELDAMeth methodology [5] the behaviors of the above mentioned *Agents* are modeled as hierarchical state machines driven by ECA (Event-Condition-Action) rules; moreover, agent interactions are enabled by multiple coordination spaces based on different coordination models (e.g. message passing, tuples, publish/subscribe). As an example, the Statecharts-based model of the behavior of the Surrogate Agent in the *CoopArch* architecture is reported in Figure 2.

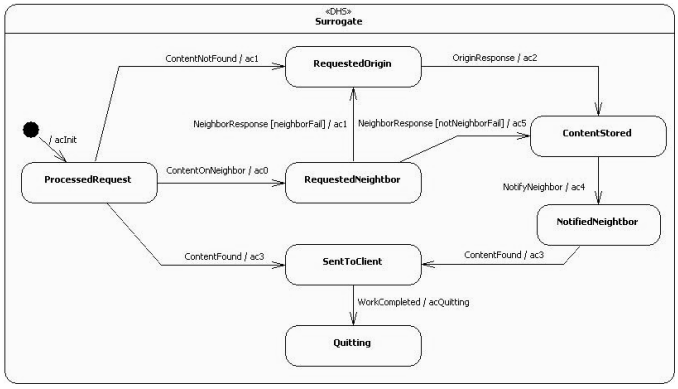


Figure 2. The Statecharts-based behavior of the Surrogate Agent in the *CoopArch* architecture.

B. Simulation parameters

The considered simulation parameters are organized in *architecture-dependent* and *architecture-independent* parameters.

The *architecture-independent* parameters are:

- The number of objects that are contained in the Origin Server (N_O). N_O is set to 100 and the objects are only considered to be static.
- The number of Surrogates (N_S), which is set in the range [2..10] to consider clusters of surrogates of different dimensions (small, medium, and large).
- The percentage of objects that are stored in a surrogate with respect to the objects stored in the origin server (C%). C% is varied from 1% to $1/N_S$ with a step of 1% for *Clustered* architecture as content cannot be replicated and from 1% to 80% with a step of 1% for *SimpleArch* and *CoopArch*.
- The average latency times among architecture components: (i) between clients and surrogates (T_{CS}), (ii) between surrogates and origin server (T_{SO}), and (iii) among surrogates (T_{SS}). As clients are very close to surrogates, surrogates of the same cluster are close to each other, and the origin is usually far away from surrogates and clients, the following relationship among the average latency times are established: $T_{SO}=3*T_{SS}=9*T_{CS}$. In the simulation runs the average latency times are set as follows: T_{SO}=90ms, T_{SS}=30ms, T_{CS}=10ms. The instantaneous values of latency times among architecture components are calculated accordingto the following link delay model [7] by setting δ_m equals to T_{CS}, T_{SO} and T_{SS} depending of the link endpoint components:

$$\delta_i = K_f \delta_m + N(K_v \delta_m, \sqrt{K_v \delta_m}) \qquad (5)$$
$$K_f + K_v = 1 \qquad K_f, K_v \geq 0$$

where δ_m is the mean delay and δ_i is the instantaneous delay for a given message. δ_i is the sum of a fixed part and a variable part. Constrains guarantee that the mean of δ_i is equal to δ_m. The variable part of δ_i is generated by a normal random variable whose mean and variance are set to $K_v\delta_m$. The distribution of the normal variable is truncated to $-K_f\delta_m$ in order to assure that δ_i cannot assume negative values. To limit the delay variability K_f is set to 0.7.

- The policy for content eviction in surrogates (EP) can be of the following types:
  - *Random*: the object to be evicted is randomly chosen.
  - *Last access*: the evicted object is the one that has not been requested for the longest time.
  - Rank: the evicted object is the less requested one.
- Average rate of client requests (λ_C) which are issued according to an exponential probability density function. λ_C is set as {0.1, 0.01, 0.001}.
- The type of distribution of the content popularity (CDP), which can be *uniform* (i.e. all the N_O objects

have the same popularity) or *Zipf* (i.e. the N_O objects are requested considering the object popularity distributed according to a *Zipf* probability density function). In particular, popularity of most popular and less popular objects is defined according to a variant of the algorithm proposed in [16] which is focused on static Web objects.

With reference to *CoopArch* there is only a specific parameter which is the number of neighbors N_N of each surrogate and is set in the range $[1..N_S-1]$.

With reference to the *Clustered architectures* (*M/SArch*, *MCArch* and *P2PArch*) the following specific parameters are defined:

- The type of distance (D_{ST}) between the surrogate originally contacted by the client and the surrogate that actually serves the client request, can be:
  - *Euclidean*, the distance is evaluated as the square root of the sum of the squares of the following distances: (i) the distance between the client and the surrogate originally contacted by the client and (ii) the distance between the originally contacted surrogate and the surrogate that actually served the client request.
  - *Manhattan*, the distance is evaluated as the sum of the distances introduced in the Euclidean case.
  - *Homogeneous*, the distance is set equal to the distance between the client and the surrogate originally contacted.
- The average latency time (T_{CAS}) between the surrogate originally contacted by the client and the surrogate that actually served the client request. The value of T_{CAS} depends on the considered type of distance (D_{ST}), thus it is set to: $\sqrt{T_{CS}^2 + T_{SS}^2}$ when D_{ST}=*Euclidean*, (T_{CS}+T_{SS}) when D_{ST}=*Manhattan*, and T_{CS} when D_{ST}=*Homogeneous*. In the simulation runs as T_{CS}=10ms and T_{SS}=30ms, T_{CAS} is equals to 31.62ms, 40ms and 30ms in the *Euclidean*, *Manhattan*, and *Homogeneous* case respectively. The instantaneous values are calculated according to the defined link delay model (see Eq. 5) by taking δ_m equals to T_{CAS}.

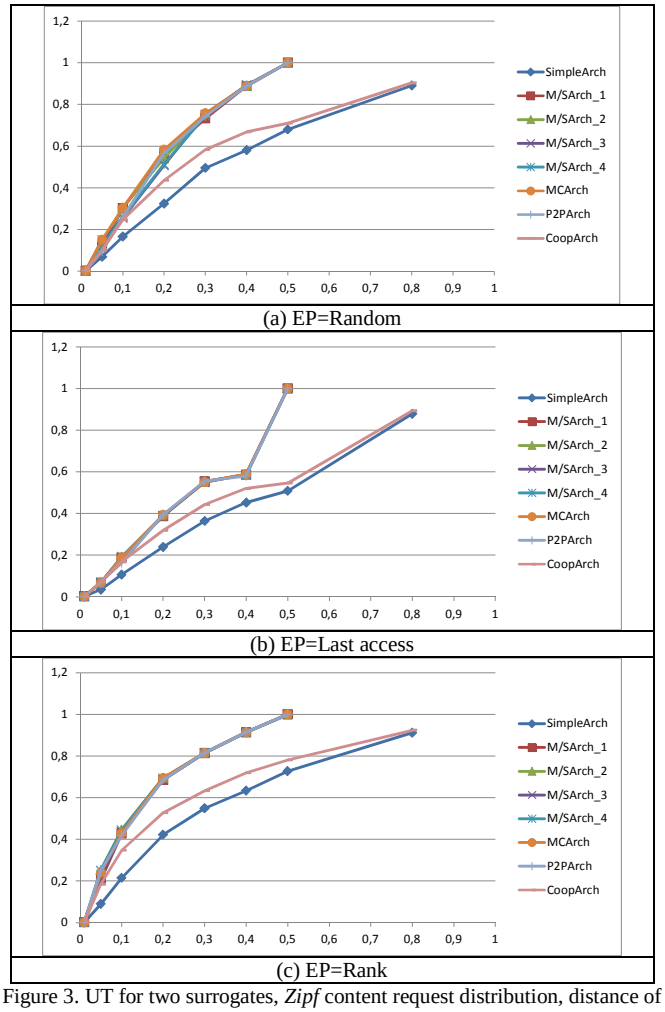
C. Simulation Results

In this Section, two sets of simulation results are presented and discussed. The first set is related to a *two-surrogate cluster* scenario; whereas the second one is related to specific choices for the content request distribution (CPD=*Zipf*) and the eviction policy (EP=*Rank*). Both scenarios refer to a *Manhattan* distance type among surrogates as this setting represents more realistic scenarios.

1) Two-surrogate cluster analysis

This analysis allowed evaluating the AUPL, CHR and UT performance indices of all the considered architectures (see Section II) in case the cluster consists of two surrogates (N_S=2). In particular, simulations are executed by setting λ_C=0.01, CPD={*uniform*, *Zipf*}, D_{ST}=*Manhattan* and by varying EP. With reference to the C% parameter, although the C% parameters is varied in the range $[1\%..1/N_S]$ (step 1%) for the *Clustered* architectures as their surrogates do not store replicated contents, C% is varied till 80% for *Non Clustered* architectures to evaluate performances in case of larger cache dimensions.

Due to space limitations only a subset of the simulation results are reported. In particular, the results for the UT, performance index with CDP=*Zipf* and the three considered Eviction Policies (*Random*, *Last Access*, *Rank*) are reported in Figure 3a-c, whereas the results for CHR and AUPL with CDP=*Zipf* and EP={*Rank*} are reported in Figure 4a and 4b respectively.



(a) EP=Random

(b) EP=Last access

(c) EP=Rank

Figure 3. UT for two surrogates, *Zipf* content request distribution, distance of the *Manhattan* type and *Random* (a), *Last Access* (b) and *Rank* (c) eviction policy.

The analysis of all the obtained simulation results led to the following important considerations:

- With CDP=*uniform* the *Clustered* architectures (*M/SArch*, *MCArch* and *P2PArch*) outperform the *SimpleArch* and *CoopArch* for the CHR and UT performance indices as the C% increases. With respect to the AUPL the architecture with higher performance

is *MCArch* if 10%<=C%<=1/N_S(=50%); only if C% is very small (<=2%) the *SimpleArch* can achieve better performance. Moreover, CHR and UT performance indices are not influenced by the adopted eviction policy whereas AUPL registers only small variations.

- With CDP=*Zipf* the *Clustered* architectures have almost the same performances with respect to CHR and UT and outperform the *SimpleArch* and *CoopArch*. However, while in the uniform case *SimpleArch* and *CoopArch* show similar performances in this case *CoopArch* performs better than *SimpleArch*. With respect to AUPL in the range [10%..50%] of C%, *MCArch* is the architecture that performs better; in case of small dimensioned cache(<2%) or large dimensioned cache (>75%) *SimpleArch* is the best performing architecture. However, the percentage of objects that are stored in a surrogate with respect to the objects stored in the origin server should not exceed given thresholds which are usually far lower than 75%.

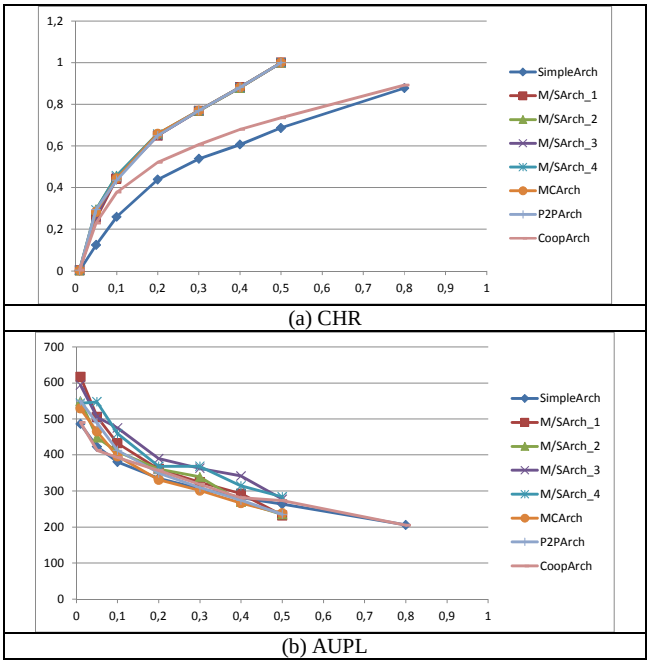


Figure 4. CHR (a) and AUPL(b) for two surrogates, *Zipf* content request distribution, distance of the *Manhattan* type and *Rank* eviction policy.

Other considerations which can be derived from the obtained results are related to the relationships between the content request distribution and the eviction policy; in particular, with a *uniform* content request distribution, the eviction policy does not affect the performance, whereas, in the case of a *Zipf* content request distribution the use of the *Random* eviction policy gives the best results for CHR and the *Rank* eviction policy provides better performance for *UT*. Finally, among the *M/SArch* architectures, the *M/SArch_2* is the one performing better.

2) Analysis of a cluster with variable number of surrogates

In this analysis the AUPL, CHR and UT performances indices have been evaluated by considering a *Zipf* content request distribution, a distance of the *Manhattan* type, a *Rank* eviction policy and by varying the number of surrogate N_S in {2, 3, 5, 10}. As among the *M/SArch* architectures the *M/SArch_2* performs better, it has been selected as the representative Master/Slave architecture.

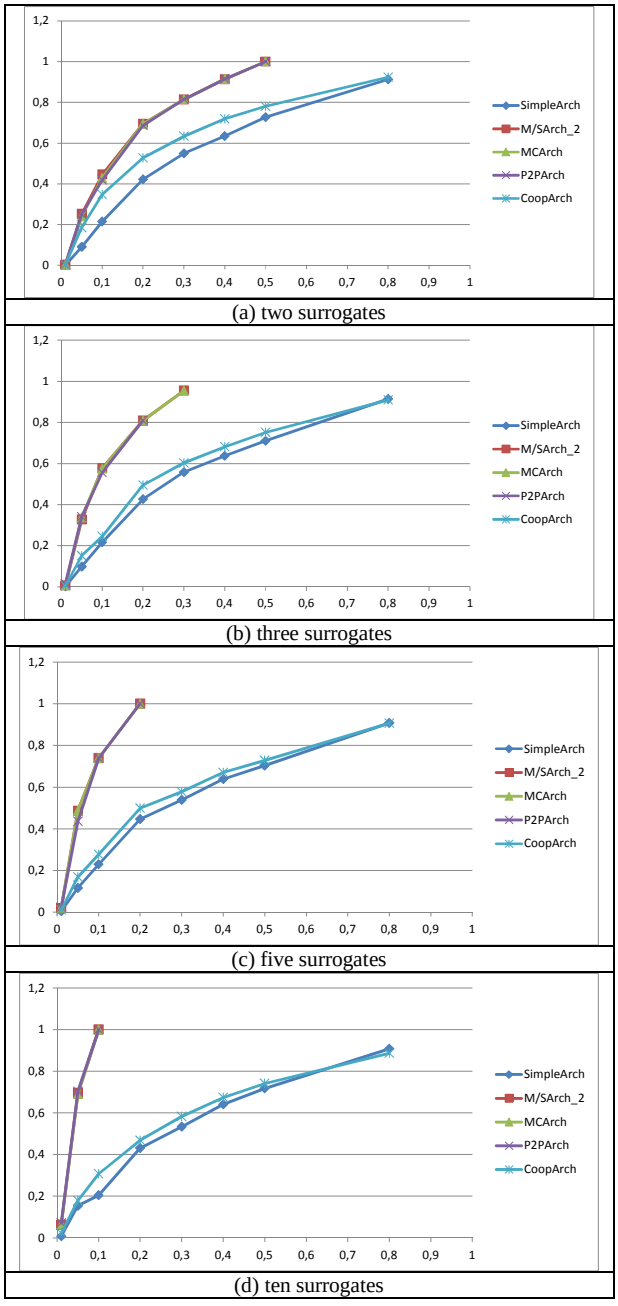


Figure 5. UT with *Zipf* content request distribution, distance of the *Manhattan* type, *Rank* eviction policy, and two (a), three (b), five (c), and ten (d) surrogates.

Due to space limitations only a subset of the simulation results are reported. In particular, the results for the UT, performance index with N_S in {2, 3, 5, 10} are reported in

Figure 5a-d, whereas the results for CHR and AUPL with N_S=10 are reported in Figure 6a and 6b respectively.

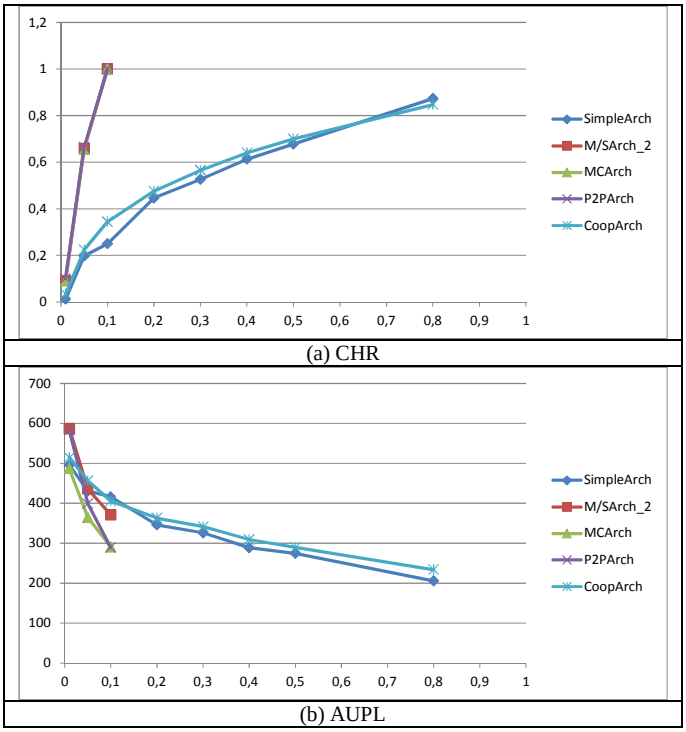


Figure 6. CHR (a) and AUPL (b) with *Zipf* content request distribution, distance of the *Manhattan* type, *Rank* eviction policy, and 10 surrogates.

On the basis of the analysis of all the obtained simulation results the following considerations can be done:

- CHR and UT of the *Clustered* architectures are better than those of the *SimpleArch* and *CoopArch* by increasing N_S. Moreover, differences in the CHR and UT performance indices between *Clustered* architectures and *Non Clustered* architectures notably augment by increasing N_S.
- Differences in the AUPL performance index between *Clustered* architectures and *Non Clustered* architectures tend to decrease by increasing N_S with reference to a C% in the range [1..1/N_S].

It is worth noting that the performance indices of the *Non Clustered* architectures may significantly improve by increasing C%. In particular, with respect to AUPL (see Figure 6b), the *Non Clustered* architectures can perform better than the *Clustered* architectures in case C% is greater than a given value: C%=68% for N_S=2, C%=52% for N_S=3, C%=52% for N_S=5 and C%=40% for N_S=10. However, cache size of a surrogate with respect to contents from a same origin server should not exceed given thresholds which are usually far lower than the aforementioned values. Finally, among the *Clustered* architectures *MCArch* performs slightly better than *P2PArch* which in turns performs better than *M/SArch_2*.

IV. CONCLUSIONS

In this paper the Agent-Based Modeling and Simulation (ABMS) approach has been exploited in the evaluation of different CDN architectures: *Conventional*, *Cooperative* and *Clustered*. Such architectures have been modeled as Multi-Agent Systems and their performances analyzed and compared on the basis of the most important quantitative performance indices for CDNs (agent user perceived latency, cache hit ratio and utility) in significant scenarios. The obtained simulation results confirm that the *Clustered* architectures outperform the *Conventional* and *Cooperative* architectures so providing useful information for the setting of the different architecture parameters. Moreover, the experimentation phase has shown the flexibility and effectiveness of the ABMS approach based on the ELDAMeth methodology for the modeling and analysis through simulation of artificial complex systems. Future research efforts are geared to develop the *Clustered* architectures atop large-scale distributed platforms such as PlanetLab [15] and MetaCDN [1] to evaluate their performances in complex and real scenarios.

REFERENCES

[1] J. Broberg, R. Buyya, and Z. Tari, “MetaCDN: Harnessing 'Storage Clouds' for high performance content delivery,” Journal of Network and Computer Applications 32(5), 1012-1022. 2009.

[2] R. Buyya, M. Pathan, and A. Vakali, “Content Delivery Networks: Principles and Paradigms,” Lecture Notes Electrical Engineering, Vol. 9, Cap. 12, Springer, Aug, 2008.

[3] M. Cossentino, G. Fortino, A. Garro, S. Mascillaro, W. Russo, “PASSIM: A Simulation-based Process for the Development of Multi-Agent Systems” in International Journal on Agent Oriented Software Engineering, special issue on Multi-Agent Systems and Simulation (M. Cossentino, G. Fortino, and W. Russo, eds), Inderscience, 2(2), pp. 132-170, 2008.

[4] G. Fortino, A. Garro, S. Mascillaro, W. Russo, “Agent-based Modeling and Simulation of Cooperative Content Distribution Networks,” in Proceedings of the 2nd Int’l Workshop on Multi-Agent Systems and Simulation (MAS&S’07 – as part of the EUROSIS European Simulation and Modeling Conference), St. Julians, Malta, 22-24 October, 2007.

[5] G. Fortino, A. Garro, S. Mascillaro, W. Russo. Modeling Multi-Agent Systems through Event-driven Lightweight DSC-based Agents. Int. J. of Agent-Oriented Software Engineering, Special Issue "Best of AT2AI-6", 4(2):113-119, 2010, Inderscience Enterprises Ltd., United Kingdom (UK).

[6] G. Fortino, A. Garro, S. Mascillaro, W. Russo, M. Vaccaro. Distributed architectures for surrogate clustering in CDNs: a simulation-based analysis. Proceedings of the 4th UPGRADE-CN Workshop on Content Management and Delivery in Large-Scale Networks jointly held with the ACM/IEEE International Symposium on High Performance Distributed Computing (HPDC'09), June 09-10, 2009, Munich, Germany.

[7] G. Fortino, C. Mastroianni, W. Russo, “A Hierarchical Control Protocol for Group-Oriented Playbacks Supported by Content Distribution Networks,” Journal of Network and Computer Applications, Elsevier 32(1), pp. 135-157, 2009.

[8] G. Fortino and W. Russo, “Using P2P, GRID and Agent Technologies for the Development of Content Distribution Networks”, In Future Generation Computer Systems. doi:10.1016/j.future.2007.06.007. 2008

[9] A. Garro and W. Russo, “easyABMS: a domain-expert oriented methodology for Agent Based Modeling and Simulation. Simulation,” Modelling Practice and Theory, Vol. 18, pp. 1453-1467, 2010, Elsevier B.V., Amsterdam, The Netherlands.

[10] J. A. Hoffer, J. F. George, and J. S. Valacich. Modern Systems Analysis and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[11] N. R Jennings, “An agent-based approach for building complex software systems,” Communications of the ACM, 44(4):35-4, 2001.

[12] J. Ni and D.H.K. Tsang. “Large-Scale Cooperative Caching and Application-Level Multicast in Multimedia Content Delivery Networks,” IEEE Communications, 43(5), pp.98-105, May, 2005.

[13] A-M.K. Pathan and R. Buyya. “A Taxonomy and Survey of Content Delivery Networks,” Technical Report, GRIDS-TR-2007-4, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Feb. 12. 2007. http://www.gridbus.org/reports/CDN-Taxonomy.pdf

[14] G. Peng, “CDN: Content Distribution Network” Technical Report TR-125, Experimental Computer Systems Lab, Stony Brook University, 2003.

[15] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir, “Experiences Building PlanetLab,” Proceedings of the Seventh Symposium on Operating System Design and Implementation (OSDI), November 2006.

[16] L. Shi, Z-M Gu, Y-C Tao, L. Wei, Y. Shi, "Modeling Web objects' popularity," In Proc. of International Conference on Machine Learning and Cybernetics, 18-21 Aug., Vol.4, pp.2320- 2324, 2005.

[17] S. Sivasubramanian, B. van Halderen, and G. Pierre. “Globule: a User-Centric Content Delivery Network.,” In Proc of the 4th International System Administration and Network Engineering Conference, Sept. 2004.

[18] K. Stamos, G. Pallis, A. Vakali, and M. D. Dikaiakos, “Evaluating the utility of content delivery networks” Proceedings of the 4th edition of the UPGRADE-CN workshop on Use of P2P, GRID and agents for the development of content networks (UPGRADE-CN '09). ACM, New York, NY, USA, 11-20. 2009.

[19] K. Stamos, G. Pallis, A. Vakali, D. Katsaros, A. Sidiropoulos, and Y. Manolopoulos, “CDNsim: A simulation tool for content distribution networks.” ACM Transactions on Modeling and Computer Simulation - TOMACS , vol. 20, no. 2, pp. 1-40, 2010.