Darmstadt University of Technology

The Quality of Availability: Tackling the Replica Placement Problem

Giwon On, Jens Schmitt, Ralf Steinmetz
{giwon,schmitt,rst}@kom.tu-darmstadt.de

KOM Technical Report 11/2001
Version 1.0
December 2001

Last update ?

Industrial Process and System Communications (KOM)

Department of Electrical Engineering & Information Technology
Merckstraße 25 • D-64283 Darmstadt • Germany

Phone: +49 6151 166150
Fax: +49 6151 166152
Email: info@KOM.tu-darmstadt.de
URL: http://www.kom.e-technik.tu-darmstadt.de/

Abstract

The importance of satisfying the availability of service and its data is becoming the most critical factor which decides the successfulness of the Internet-based services and applications. In this work, we take the availability issue into account as a QoS focus, and evaluate the achieved quality of a service by answering the question ‘how much the service satisfies the required availability?’. We introduce the concept of quality of availability (*QoA*) in which the availability is treated as a new controllable QoS parameter, and study the QoA by tackling the replica placement (RP) problem. We distinguish the RP problem in three questions, i.e., (1) finding a ‘good’ placement with fixed number of replicas, (2) checking the achieved QoA with a selected replica placement, and (3) determining the number and location of replicas for satisfying a required QoA. For each RP question, we reviewed some problem solving algorithms from heuristic and exact methods. We built an experimental simulation environment and implemented the algorithms such as *TransitNode* heuristic and exact *state enumeration* method. Our primary result shows that (1) the location of replica is a relevant factor for availability QoS, (2) the heuristic method is good for large-size graphs, but does not give any availability QoS guarantee and requires a high replica degree, and (3) the exact methods guarantee the required availability QoS, although their run-time overhead is very high. We currently extend the simulation model by adding QoS and consistency components into the model.

Keywords

quality of availability (QoA), availability, redundancy, replication, replica placement problem, heuristics, exact methods

1 Introduction

1.1 From Satisfying Time-Constraint Requirements to Satisfying Availability Requirement

The rapid popularization of Internet-based distributed services and applications have inspired the research and development of technologies for providing resource assurance and service differentiation which have motivated the development of the concept, architecture and mechanisms of *quality of service* (QoS). The main focus of QoS is improving the service quality by addressing the issues of resource management, service differentiation and performance optimization [1,2].

Even though there are many significant efforts, results, technology advances and solutions in QoS since the last 20 years, their application to the commercial products or to systems was not so successful, in comparison to their attention in research field. One critical reason is that, as H. Schluzrinne[3] also pointed out in his invited talk at IWQoS’01, the main research focus for QoS was to solve the time-constrained problems. This is because the applications on the Internet, such as video conference, video-on-demand, tele-teaching/learning, Internet telephony, which continuously and strongly motivated the development of QoS technologies, have the characteristics of using multimedia, i.e., having requirements related to real-time constraints (digital video streaming). Indeed, the availability issue has not been mentioned so far, and there were no significant works with the focus on how to satisfy the availability requirement.

The importance of satisfying the availability of service and its data arises recently more and more and is becoming the most critical factor which decides the successfulness of the Internet-based services and applications. An interview statistic [4] also reflects the importance of satisfying the availability requirement of end-users for a successful service and application.

1.2 From Providing One Availability Level (‘Five Nines’ for Everyone and Everytime) to Differentiating Availability Levels

Observation1 - high availability (HA) technology trend[5]: the focus of HA research efforts is mainly towards achieving the so-called ‘five nines’(99.999%)

Observation2 - developing replication mechanisms for increasing availability of services and their data in a distributed multimedia system [6]: we identified that not all service operations and not all data access functions require the availability level of the ‘five nines’.

On the other side, not all system components which all together build the whole service system do not require the same availability level. The availability level that can be required by individual system components depends on the fact of how much they should be reliable and critical for offering the service.

1.3 Resulting Idea

The work in this paper is strongly motivated from the two facts discussed above, and will especially take the availability issue into account as a QoS focus. So when we evaluate the achieved quality of a service, we first check, (a) ***how much the service satisfies the required availability***, and then (b) how much the service meets the required time-constraints. We pack all the issues and problem solving mechanisms which are related to the question (a) into a concept which call ***Quality of Availability (QoA).***

2 The Concept of Quality of Availability (QoA)

The basic idea of the QoA concept is that the availability can be defined as *a new controllable, observable QoS parameter*. Indeed, we move the focus of the objective functions for the resource and performance optimization problems of the QoS field from satisfying the time-constraint requirements such as minimizing transmission delay, jitter, and/or synchronization to satisfying the availability requirements such as minimizing failure time of service systems and their components and maximizing the total time amount in which the required service functions as expected and its data are reachable.

2.1 What are the Relevant QoA Parameters?

A. Required service availability: this parameter can be specified by the service consumers and the underlying applications. For this point, we need the function (definition) for specifying service availability. By analysing service scenarios and data characteristics of target applications, we can better specify the required service availability. Quantifying the availability requirement values for individual data types presented in the replication work for medianode[6] can be an approach for gaining this parameter.

B. Supplied availability: it can be offered by the corresponding service system (components and its data). For this point, we at first have to define service system, its system components, its data and then the availability value offered. We have to clearly define how the offered availability (value, level) can be formulated, i.e., from each system component, its sub-component, data, etc. For this purpose, we need to have an availability model with respect to the system component's depth/level, its redundancy degree, and so on.

C. Failure probability of the service system and its components inclusive data managed: Because of the failure probability is defined as [*1 - AvailabilityProbability*], we often specify either availability probability or failure probability.

D. Redundancy Degree (is replication degree analogous to the redundancy degree?): deciding the redundancy degree depends not only on the required service availability level, but also on the acceptable cost limit fixed by service/content provider. The service consumers often want to receive a service that is highly available, if possible, while the resources for which the service/content providers should pay are limited.

E. Orthogonality to other QoS parameters: beyond the QoA parameters mentioned above, we could take the typical QoS parameters like delay, jitter and data loss. Especially, taking the delay as QoS parameter into account makes the QoA model more realistic, because the service consumers usually expect to receive a service within an acceptable response time, when they require the service, and when it is assumed that the service is available. One another QoA parameter can be the consistency which can be modelled either as staleness or the total time duration how long the data were inconsistent.

In our current simulation model (the base model), the parameters in E are not addressed.

2.2 Why to tackle the replica placement problem?

Replication is the proven concept for increasing the availability for distributed systems. Replicating service (components) and data from the origin system to multiple networked computers increases the redundancy of the target service system, the system components and the data managed by the system, and indeed the availability of the service is increased.
The important decisions what replication management system concerns are:
(a) what to replicate? - *replica selection problem*,
(b) where to place the replica? - *replica placement problem*, and
(c) when and how to update them? - *update control problem*

From these decision problems, we especially study the replica placement problem, because the placement problem depends mostly on the available, but continuously changing system/network resources and on the costs what the service/content providers should pay for satisfying the required availability.

3 The Replica Placement (RP) Problem

3.1 Notations

Before we start to model the RP problem, we check all notations in Table1, which are used for formulating and specifying the RP problem throughout this work.

Notation	Description
$G(V, E)$	a graph with V, the set of nodes, and $E \subset V \times V$, the set of links
D	a set of service demanding nodes and a subset of nodes of G, $D \subset V$
S	a set of service supply nodes and a subset of nodes of G, $S \subset V$
R	a set of replica nodes, a subset of nodes of G, $R \subset S \wedge R \subset V$
$\|V\|, \|D\|, \|S\|, \|R\|$	the cardinality of the node sets V, D, S, R
$T(R)$	a topological placement of R, i.e., the location of R's elements
p_e	the availability probability of the element e ($e \in V$ or $e \in E$) for service supply
q_e	the failure probability of the element e ($e \in V$ or $e \in E$) for service supply: $q_e = 1 - p_e$
pd_e	the required service availability of the element e ($e \in V$) for service demanding node
x_e	a boolean variable for the state of the component e ($e \in V$): 1 if e functions else 0
$a(G)$	availability of G
g_i	a state of G
$G(g_i)$	partial graph of G associated with g_i
$A_e(R)$	the achieved QoA for a given service demanding node e of V with a replica set R: 1 if satisfied else 0
$\bar{A}(R)$	the achieved QoA for all service demanding nodes of G with a replica set R: 1 if satisfied else 0

Table 1: Notation.

3.2 Problem Formulation

Distributed, networked service systems that consist of storage/server nodes and network connections between them can be modelled as a graph, *G(V,E)*, where V is the set of nodes and E the set of connection links. This graph is *static* if the member and the cardinality of V and E do not change else *dynamic*. The graph is said to be *stochastic* when each node and link are parameterized, statistically independently on another, with known failure or availability probabilities.

The RP problem can be distinguished between the constrained RP (CRP) and unconstrained RP (URP): In the CRP, there are a set of service demanding nodes *D* and a set of service supply nodes *S*, so that $D \cup S = V$ and $D \cap S = 0$, and the replica set R can only be built from the nodes of the set S, $R \subset S$. In the URP, every node can be either a service demanding node or a service supply node,i.e., there is no *D* and *S*, and $R \subset V$.

Concerning to the RP problem, the following sections introduce three questions that are explored in this work. For all three questions, we model the RP problem as a *static, stochastic and unconstrained* graph.

3.3 Finding a 'Good' Placement with Fixed Number of Replica

Problem Statement: As input, a stochastic graph and a positive integer number k are given. The objective of this problem is to place the *k* replicas on the nodes of *V*, i.e., find *R*, *|R| = k*, and *T(R)*, such that a given optimization condition *O(|R|, T(R), a)* is satisfied for given availability requirement for service demanding nodes. How well the optimization condition is satisfied depends on the size of R and the topological placement *T(R)*. Because of the main goal associated with placing replicas on given networks in our work is satisfying/improving availability QoS, we take the availability and failure parameters as our key optimization condition, *O(|R|, T(R), ReachedAvailability)*. With the use of maximum, 90%-tile, and mean clients' required availability value, the optimization condition can be denoted as *O(|R|, T(R), 1), O(|R|, T(R), .90), O(|R|, T(R), avgA)* respectively. For these conditions, the set *R* of replicas must be chosen such that the maximum and/ or average failure bound for service (and its data) access is met the requests from any demanding node (client node) of V.

Algorithms: The RP problem can be classified as NP-hard discrete location problem. In literature, many similar location problems are introduced and algorithms are proposed to solve the problems in this category. The heuristics such as *Greedy, TransitNode, Vertex substitution, Tabu Search*, etc. [7,8,9,10] are applied to many location problems and have shown their efficiency. In this work, we reviewed the *TransitNode* heuristic. The basic principle of the *TransitNode* heuristic is that nodes with the highest (in/out) degrees, i.e., the number of connection links to adjacent nodes, can potentially reach more nodes with smaller latency. So we place replicas on nodes of V in decending order of (in/out) degree. This is due to the observation that nodes in the core of the Internet that act as transit points will have the highest (in/out) degrees [11]. Figure 1 illustates a *TransitNode* algorithm.

.

```
Algorithm TransitNode
Input:      a parameterized graph G (V,E) and a number k;
Output:     a replica node set R
Initialize variables:replica node set R = candidate node set = {}
1. calculate max/min/mean values for
2.      supply availability probability,
3.      failure probability, and
4.      (in/out) degree
5. forall nodes v of V
6.      if (availability value of v is greater than the mean value &
7.       its degree is greater than the mean degree) then
8.              put v into a candidate replica node set candSet
9. while (|R| <= k)
10.     take the best k nodes from candSet and put them into R
11.return R
```

Figure 1: The *TransitNode* algorithm for finding a placement

For taking the best k nodes from the candidate node set at step 9, we check whether the candidate node is an adjacent node of any already selected replica node of R. If yes, the test node is not selected as replica node. Instead of that node, the next candidate node is taken from the candSet and tested for the selection. This test is for guaranteeing that no replica nodes selected are adjacent to each other. Note that this algorithm does not deal with the check whether and how to guarantee the required availability with the replica set R during the replica selection phase.

Example: We applied the *TransitNode* algorithm to an example stochastic graph which represents a model of a virtual network topology similar to the Internet topology (Figure 2). In this example graph G(V,E) with |V|=10 and |E|=20, nodes and links are parameterized with randomly generated probability values, e.g., the demanding and supply availability values of nodes are between 90% and 99%, and the failure probability values of links are between 1% and 10%. For a given number $k = 1$, the candidate replica nodes are {6}, {3}, {4}, {8} and {9}. By considering only the high degree, the algorithm decides {6} as the replica set. In other case, when the supply availability value is only taken into account, {9} will be decided as the target replica set instead of {6}.

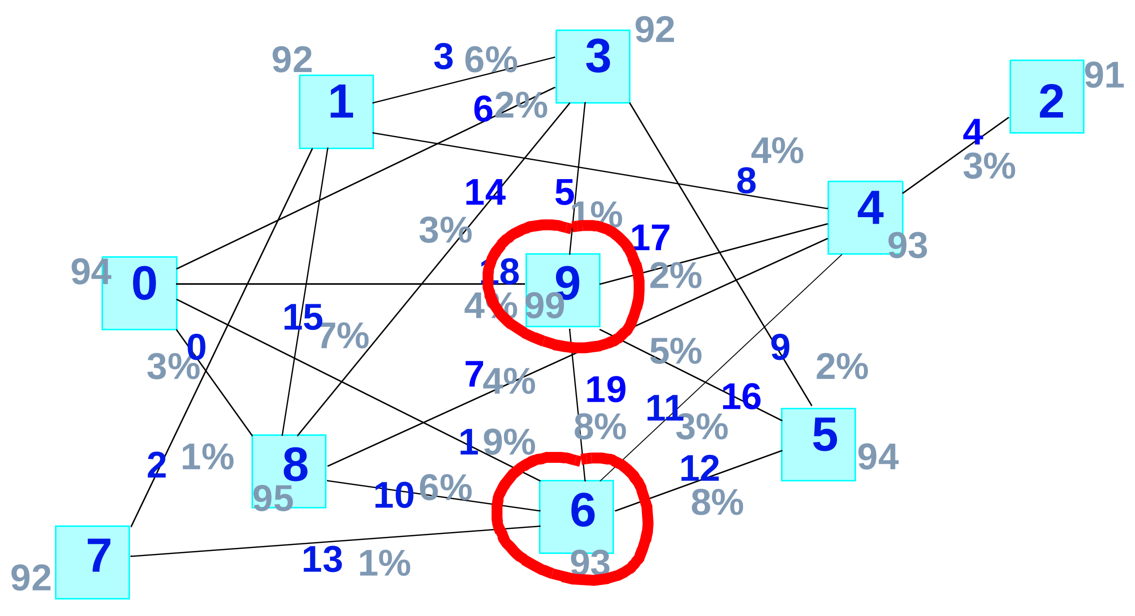

Figure 2: An example stochastic graph.

Analysis: Since the common goals associated with placement problems are reducing clients' download time and alleviating server load, like in [12, 13], the main feature of the problem solving approaches for this problem category is that they usually addressed the cost and resource minimization issues, but not the question how to guarantee the required availability. Furthermore, we can find an 'good' upper bound, if the selected placement meets the required availability QoS, but it is not guaranteed that the selected placement always meets the availability requirement.

3.4 Calculating the Reached QoA

Problem Statement: Given a stochastic graph G and a replica set R with its topological placement T(R). The objective of this problem is to check for all demanding nodes whether the reached availability satisfies the required QoA for them, i.e., whether $\bar{A}(R)$ is 1 or 0. In compare to the problem of finding a good placement handled in Section 3.3, this problem requires a solution which exactly tests whether the result is 1 or 0. Some similar works are introduced in the literature, which are devoted to the problem of *network reliability*[14]. And, the methods that provide an exact reliability are called exact methods, in opposition to the heuristic methods which provide approximate result.

Algorithms: Enumerating all possibilities without skipping any solution case requires to take exact methods for solving this problem. From some exact methods which are proposed in the literature, we adopted the state enumeration method[15] and modified for our problem. In the state enumeration method (shortly, SEM), the state of each node and each edge are enumerated: the state value is either *1* when it functions or *0* when it fails. Indeed, there are *2**(|V|+|E|)* states for a graph *G = (V,E)*, i.e., *2**(|V+|E|)* partial graphs for the G. We then check the QoA for all partial graphs with the replica set given as input. Figure 3 illustates a *StateEnumeration* algorithm.

.

```
Algorithm StateEnumeration
Input:      a parameterized graph G (V,E) and a placement (replica set R);
Output:     the QoA: either 1 (satisfied) or 0 (not satisfied)
Initialize variables:state matrix
1.    for all nodes v of V/R
2.    build a state martix for all partial graphs of G
3.    for all states
4.     check the availability of the state, i.e., exist at least a path?
5.    for the states for which the checked availability value is 1,
6.     compute availability probability for each state
7.    sum all the availability probability values of satisfied states
8.    if p_a(v) <= A_v(G) then A_v(R) =1
9.    for all nodes v,
10.    if A_v(R) =1
11.    then dach A(R) = 1, i.e., the QoA of G with R is satisfied
12.    else dach A(R) = 0
13.   return QoA
```

Figure 3: The *StateEnumeration* algorithm for calculating the reached availability, QoA

Example: We applied the *StateEnumeration* algorithm to an example stochastic graph G(V,E) with |V|=5 and |E|=6, placement R={2}, |R|=1, as shown in Figure 4. As test node, we take the node 0 which has the availability requirement value 97%. Table 2 presents the state matrix, the availability and the sum of availability probability value for each partial graph. In this example, we only encounter the states of nodes to reduce the time complexity, i.e., from *2**(|V|+|E|)* to *2**|V|*.

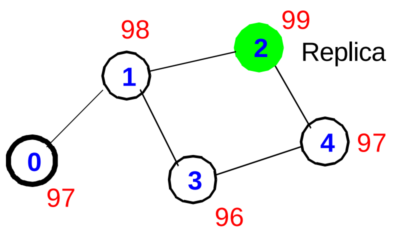

Figure 4: An example graph G=(V,E), |V|=5, |E|=6.

The first column means number of the partial graphs to be tested. Instead of considering all partial graph cases of *2**(|V|+|E|)*, we only have the half size of them by skipping the cases in which the node 1 is 'not available', i.e., (the state value of the node 1 is 0), because the node state 'zero' of the node 1 causes no further possible connection for the test node 0 to build any path to the replica node 2. After building a state matrix for the nodes 1,2,3 and 4 at the second column, we check whether there is any path between the node 0 (test node) and the node 2 (replica node) at the third column. According to result of this check, we calculate the availability probability values for each partial graph of which the availability check value is 1: as shown at the forth column, we calculated the *Pr(G_m)* just for the first 4 partial graphs (m = 1,2,3 and 4). The summation of the availability probability values of the four satisfied states is: *A_node0(G) = Pr(G_1) + Pr(G_2) + Pr(G_3) + Pr(G_4) = 0,97135624*, and this availability value is greater than the availability value required by the node 0. Indeed, the QoA for the node 0 is: *A_node0({2}) = 1*.

m	StateMatrix	a(G_m)	Pr(G_m)
1	1 1 1 1	1	0,98 * 0,99 * 0,96 * 0,97
2	1 1 1 0	1	0,98 * 0,99 * 0,96 * 0,03
3	1 1 0 1	1	0,98 * 0,99 * 0,04 * 0,97
4	1 1 0 0	1	0,98 * 0,99 * 0,04 * 0,03
5	1 0 1 1	0	/
6	1 0 1 0	0	/
7	1 0 0 1	0	/
8	1 0 0 0	0	/

Table 2: Result of calculating the reached availability for the node 0.

3.5 Determining the Number & the Placement for Satisfying Required QoA

Problem Statement: As input, only a stochastic graph G is given. It has to be determined (a) how many replicas must be deployed and (b) where these replicas should be placed to guarantee the required QoA?.

Algorithms: Satisfying a certain, required QoA value with a guarantee means that we have to offer always a replica set which fulfils the given QoA requirements in any case. Heuristics are not proper approaches for solving this problem, because they give not always a solution with QoA guarantee. To solve this problem, we generally can use the exact methods, one possible case may be enumeration method which is described in the subsection 3.4. For solving this third problem, we only need to call the state enumeration method *2**|V|* times, i.e., for all the possible replica solution sets. The algorithm complexity is then O(*2**|V| * 2**(|V|+|E|))*.

4 Experimental Setup for Simulation

Model Components and Their Parameters: We built an experimentation environment to perform a simulation study for the three RP problems addressed in the Section 3. We implemented a *StateEnumeration* and a *TransitNode* algorithms for solving the addressed three problems with the example graph presented in Figure 2.

The Availability and failure probability parameters for service demand nodes, service supply nodes and edges between nodes are of single values. We have not decoupled the availability values between the demanding and supply nodes, i.e., the values for these two types of nodes are the same. The role, whether demanding or supply, is only differentiated, when a node is selected as a replica node.

As replication model, we assume the *full replication* in which the whole data items of an origin server system are replicated to some nodes. Mirroring is a typical case of this replication model.

Furthermore, we consider an unconstrained placement model in which all nodes are the potential candidate to be selected as replica node.

The model components which are ignored at the current simulation model, but planned to be added for building a more realistic simulation environment are the QoS model, consistency model, service (data) access model, and communication model.

Used Libraries and Platforms: The simulation program is written in C/C++ and runs on Linux machines. We used Leda graphic library [16] to create and manipulate the test graphs. The input/ output graph files can be generated either by specifying the graph parameter at run time, or by reading the files which are generated by the topology generator Tiers[17].

5 Results and Evaluation

Table 3 shows the result from the state enumeration algorithm. We started the program with a replica degree 1, i.e., $k=|R| = 1$, and selected each node as replica node. The second row of Table 3 shows the achieved QoA at each node: a replica placement on the node 9 satisfies the required QoA for all nodes. The third row gives the achieved availability in percentage. From these values we can predict that the node 8 with 80% of the achieved availability could also be a good candidate for being a replica node. The fourth and fifth rows show the reached availability values in the ratio for the nodes for which the QoA was not satisfied and satisfied, respectively.

Node	0	1	2	3	4	5	6	7	8	9
QoA	0	0	0	0	0	0	0	0	0	**1**
%QoA	0.6	0.3	0.1	0.2	0.6	0.6	0.5	0.2	0.8	**1**
avgAf	0.97489	0.96766	0.76755	0.97103	0.97425	0.97486	0.96950	0.95407	0.96523	x
avgAs	1.01619	1	x	1	1.00976	1.01517	1.00815	1	1.02232	1.05682

Table 3: A test result from the *StateEnumeration* method for the example graph in Figure 2

Table 4 presents a test result of the *TransitNode* algorithm presented in Figure 1. After running the *TransitNode* program several times, we could come to the placement which satisfies the required QoA for all nodes: *k* = |*R*| = *2 and R={3,4}*.

Node	3,4
QoA	**1**
%QoA	**1**
avgAf	x
avgAs	1.05434

Table 4: A test result from the *TransitNode* method satisfying the required QoA for the example graph in Figure 2

The following observations can easily be identified from our first result: (1) the location of replicas is a relevant factor for the availability QoS; (2) using the heuristic method is more efficient than the exact method, at least in terms of the runtime complexity, to find a good placement for large graphs. But, the replica degree (the number of replicas) of their placement results are in most cases higher than those of exact methods. Furthermore, the heuristics give no guarantee for availability QoA; (3) in opposite to the heuristic method, the exact method guarantees the availability QoS with its placement results, although the runtime complexity is very high: *O((|V|-|R|)*2**(|V|+|E|))* and *O(2**|V|*(|V|-|R|)*2**(|V|+|E|))* for the availability checking and the guaranteed QoA problems, respectively.

Some techniques and ideas which reduce the runtime complexity and improve the current simulation methods are:

- decomposition and parallelization of calculation modules of the exact method used: to reduce the runtime complexity, we first decompose the whole possible replica sets (the whole partial graphs of a given graph) in a number of replica subsets and then run the exact method for each portion of replica subsets in parallel.
- development of other advanced problem solving methods
- combination of heuristics and exact methods: by using heuristics, we find an upper bound for all possible placements which satisfy the required QoA, and then use exact method to fix the final placement.

6 Model Extension

The current simulation model should be extended to build a more realistic QoA model. The extensions which are currently performed are described in the next subsections.

6.1 QoS Model

1. We address the delay as a QoS parameter. This is because of the fact that the achieved QoA is then in a more realistic sense, when the required service and the corresponding data is accessible in a certain response time which is set either by applications or the users.

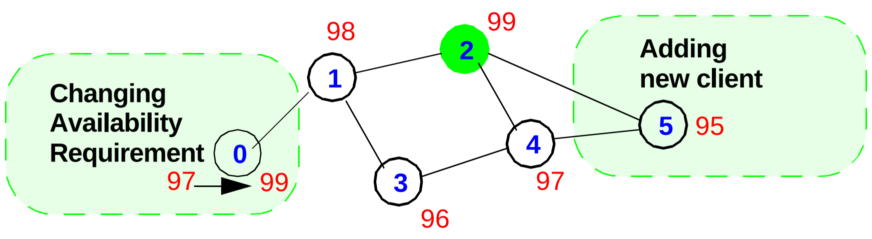

Figure 5: modeling the admission control concept

2. As further QoS extension, we also address a support of the admission control concept. As shown in Figure 5, we can put some service usage scenarios into the graph: the first scenario is changing the availability requirement of any demanding node during the service time, and the second scenario is adding new demanding nodes into the graph. In all cases, we check whether the selected placement still fulfils the QoA for all nodes, and if not, determine a new placement. We can model the admission control concept by using dynamically changing stochastic graph.

6.2 Consistency Model

Keeping the consistency between replicas is also an important issue. Consistency can be defined as several forms. Some possible definitions are: (1) the number of accesses to replicas which are of stale state, or (2) the time duration how long the replicas are inconsistent.

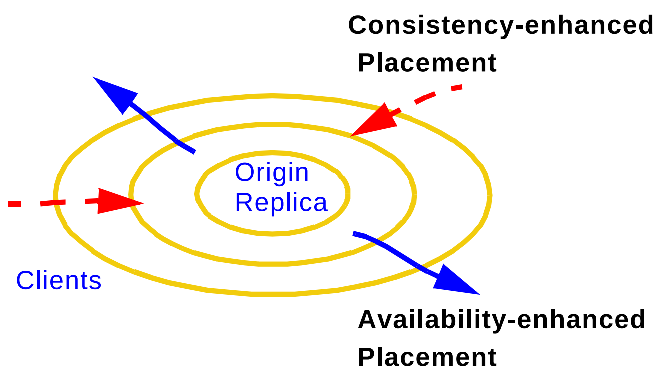

Figure 6: Consistency and availability enhanced placements

By considering both the availability and consistency issues, finding an absolute best placement is more difficult, because the placement for increasing availability will be given in a form that the replicas are placed as near as possible to client nodes, while the placement for enhancing consistency will be given in the form of ‘server-centric’ to minimize the cost needed for controlling the consistency which is usually done by exchanging the update related messages between replica nodes. Figure 6 illustrates the two placement policies.

The consistency component can be modelled either (1) by setting the maximal number of replicas, or (2) by fixing the maximal radius size of replica set, i.e., by fixing the largest hop size between replicas, or (3) by checking the availability that should be satisfied for all selected replicas.

7 Summary

In this work, we addressed the service and data availability issue. We handled availability as a new controllerable QoS parameter. We especially tackled the replica placement problem and specified three different placement problems: (1) finding a relatively good placement, (2) checking the achieved availability QoS, and (3) determining the number and the location of replicas to satisfy required availability QoS. We built an experimental simulation environment to perform a simulation study with exact and heuristic methods. We implemented an *StateEnumeration* and a *TransitNode* algorithms to solve the placement problems. The primary result shows already that the location of replicas is a relevant factor for the availability QoS.

We are currently extending the basis QoA model presented in this work to build a simulation model for studying the proposed QoA concept in a more realistic sense.

References

[1] Zheng Wang. *Internet QoS: Architectures and Mechanisms for Quality of Service*. Lucent Technologies, 2001.

[2] J. Schmitt. *Heterogeneous Network QoS Systems*. PhD thesis, Darmstadt University of Technology, December 2000.

[3] H. Schulzrinne. “QoS over 20 Years”. Invited Talk in *IWQoS’01*. Karlsruhe, Germany, 2001.

[4] SEQUIN project. “Quality of Service Definition” *SEQUIN Deliverable D2.1*, April 2001. (http://www.dante.net/sequin/QoS-def-Apr01.pdf)

[5] HP Forum. *Providing Open Architecture High Availability Solutions*, Revision 1.0, Feb. 2001. (http://www.mcg.mot.com/us/products/solutions/ha_solutions.pdf)

[[6] G. On, J. Schmitt and R. Steinmetz. “Design and Implementation of a QoS-aware Replication Mechanism for a Distributed Multimedia System”, in *Lecture Notes in Computer Science 2158* (IDMS 2001), pp.38-49, Sep. 2001.

[7] N. Mladenovic, M. Labbe and P. Hansen: “Solving the p-Center Problem with Tabu Search and Variable Neighbourhood Search”, July 2000. (paper available via http://www.crt.umontreal.ca/)

[8] Metaheuristics - global optimization (paper)

[9] Christos H. Papadimitrio and Kenneth Steiglitz. *Combinatorial Optimizatioon: Algorithms and Complexity*. Prentice-Hall, ISBN 0-13-152462-3, 1982.

[10] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, ISBN 0-201-05594-5, 1984.

[11] Pierre, I. Kuz, M. van Steen and A.S. Tanenbaum. “Differentiated Strategies for Replicating Web documents”, In *Proc. of 5th International Workshop on Web Caching and Content Delivery*, Lisbon, May 2000.

[12] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt and L. Zhang. “On the Placement of Internet Instrumentation”, In *Proc. of IEEE INFOCOM’00*, Mar. 2000.

[13] S. Jamin, C. Jin, A. R. Kurc, D. Raz, Y. Shavitt. “Constrained Mirror Placement on the Internet”, In *Proc. of IEEE INFOCOM’01*, pp. 31-40, 2001.

[14] M. Nicola and M. Jarke. “Performance Modelling of Distributed and Replicated Databases”, *IEEE Trans. on Knowledge and Data Engineering*, Vol.12(4), pp.645-672, July/Aug. 2000.

[15] C. Lucet and J.-F. Manouvrier. “Exact Methods to compute Network Reliability”. In *Proc. of 1st International Conf. on Mathematical Methods in Reliability*, Bucharest, Roumanie, Sep. 1997. (paper available via http://ramp.ucsd.edu/resources.html)

[16] Leda - the library of efficient data types and algorithms. Version 3.8. Algorithmic Solutions Software GmbH. (http://www.algorithmic-solutions.com/as_html/products/products.html)

[17] Tiers topology generator, available via http://www.isi.edu/nsnam/dist/topogen/tiers1.0.tar.gz