Self-stabilizing Quorum Systems for Reliable Document Access in Fully Distributed Information Systems

Fatima Belkouch, Marc Bui and Liming Chen

Université de Technologie de Compiegne
Département Genie Informatique
60205 Compiegne Cedex
FRANCE
e-mails: {fatima.belkouch, marc.bui, liming.chen}@utc.fr

Abstract: The fast development of a worldly distributed information, such as the World Wide Web (WWW), is one of the major evolutions of present computer technology. Within the TransDoc project, which aims at providing intelligent vehicles for document access in a fully distributed information system such as Internet, we study the self-stabilizing construction of quorums in order to ensure reliable document access. Our approach consists of match-making the quorum systems with the self-stabilization paradigm in order to ensure mutual coherence of document copies and to guarantee some fault tolerant behavior. In this paper two self-stabilizing algorithms for constructing quorums are proposed. Performance evaluation driven with simulations and a Markov chains modelling, show that such constructions improve the quorums quality as the self-stabilizing quorums, given their convergence property, automatically tolerate transient faults and dynamic changes of the network.

Keywords: Self-Stabilization, Distributed Systems, Quorum Systems, Coteries, Spanning Tree

1. Introduction

The fast development of a worldly distributed multimedia information, such as the World Wide Web (WWW), is one of the major evolutions of present computer technology. In such a context, the underlying communication network has a dynamic behavior. Sites may disappear because of site or link failures, while new sites may be introduced because of their insertion. For the sake of performance and reliability, the same document may be duplicated on several sites, for instance in proxies and mirror sites. Even though access of a document for reading is much more frequent than access for writing, it is necessary to ensure a mutual consistency between document copies, the "one-copy equivalence" property [Ray92], and the availability of documents when failures of sites or links occur.

Within the TransDoc project [CDF97], [Chen98], which aims at providing intelligent research vehicles to facilitate multimedia document access in a fully distributed information system such as WWW, we study the issue of reliable document access in a dynamic communication network in which failures of sites or links may occur [BBBC97], [BC97]. Our approach consists of match-making the quorum systems with the self-stabilization paradigm [Sch93] in order to maintain document consistency while ensuring some fault tolerant behavior of the system.

Quorum systems are taken as a basic tool for providing a uniform and reliable way to achieving co-ordination in a distributed system [Ray92], [Mae85], [AA91]. On the other hand, the self-stabilization paradigm aims at designing distributed algorithms with the ability of recovering spontaneously from any arbitrary state of the system, without any sort of an intervention from outside [Sch93]. So, a self-stabilizing system does not require any initialization, tolerates transient faults and adapts itself to the dynamic evolution of the network. The self-stabilization property is very useful for quorum systems in such a context as that where sites may be inserted or crashed and then recovered spontaneously from an arbitrary state. When the intermediate period between two successive network topology changes, or the time in between a recovery and the next crash is long enough the system stabilizes and the current quorum system is considered as available. Our contribution in this paper is the design of self-stabilizing quorum systems for documents access [BBBC97], [BC97]. The basic idea of constructing such quorum systems consists of building some self-stabilizing logical structure over the network and of computing quorums using a topology dependent function. Our constructions [BBBC97], [BC97] are both based on a self-stabilizing spanning tree. The first one uses a composition function of quorums and gives a self-stabilizing arborescent quorum system. The second one, which improves our first construction, derives a logical triangle lattice structure from the spanning tree, and defines a self-stabilizing planar quorum system.

The dynamic system is subject to frequent topology changes. Self-stabilization protocols

for such a system should stabilize in between two successive topology changes. Thus, *the stabilization time T* is an important parameter in the design of dynamic self-stabilizing algorithms. Beside the classical criteria on quorum systems such as quorum size, the performance of the self-stabilizing quorum systems is also evaluated in terms of self-stabilization time T, which is expressed by the number of rounds the system takes to converge to a state where a quorum system is correct. We have evaluated, under a sequential demon, that $O(N^2)$ rounds are necessary for the quorum system stabilization. Furthermore, the stabilization cost introduced by this new parameter is estimated on the assumption of a stochastic model. The stabilization time measures would rather quantify the quorum system *reliability,* which is an appropriate and important measure of performance, than its *availability.*

The remainder of the paper is organized as follows : Section 2 gives some necessary definitions in the quorum theory and some self-stabilizing paradigms. Section 3 presents our two self-stabilizing quorum constructions. Section 4 defines the legitimate states and proves the closure and convergence properties of our two previous algorithms. Section 5 assesses the stabilization time, details the quorum evaluation performance based on Markov chains, and presents some simulation results. Section 6 compares our results to other related work and gives some concluding remarks.

2. Quorum Systems and Self-stabilization

Early work on quorum systems used voting to define the quorums [Tho79]. The simple majority system is an example of a voting system [Gif79]. [MB85] and [Mae85] related quorums to intersecting set systems and defined *coteries* and the concept of *non-domination.* Alternative protocols based on quorum systems appear in [AA91] [Kum91].

Quorum systems have been used in the study of distributed control and management problems such as *mutual exclusion* [Ray86], *data replicated protocols* [SBDS85], *name servers* [MV88], *selective dissemination of information* [YGM87] and *distributed access control and signatures* [NW96].

Generally, a system based on a quorum concept works as follows : to perform some action, the user selects in a transparent way a quorum and accesses all its elements. The intersection property guarantees that the user has a consistent view of the current state of the system.

2.1 Quorums, Coteries and NDQ

Let us define the basic terminology to be used in the paper.

A *quorum system* $Q = [q_1, q_2, \ldots, q_k]$ over a set V is a non-empty set where each element is a non-empty subset of V and $q_i \cap q_j \neq \phi$, $i, j \in \{1, \ldots, k\}$.

A *coterie* C over set V is a quorum system over V which is minimal under set inclusion. There are no $q_1, q_2 \in C$ such that $q^1 \subset q^2$.

A quorum system Q *dominates* a quorum system Q' if $Q \neq Q'$ and if for every $q' \in Q'$ there exists $q \in Q$ such that $q \subseteq q'$. This suggests the definition of a *non-dominated quorum system*, or NDQ for short: a quorum system is non-dominated if there is no other quorum system that dominates it.

2.2 Performance Criteria for Quorum Systems

The performance of the quorum systems may be evaluated according to several criteria : *quorum size* [Mae85], *load* [PW95] [NW94], *availability* [Tho79], [NW94] and *failure cost* [Kum91; Baz96]. As these criteria are generally conflicting, there is no quorum system construction that might be optimal with respect to all of them.

- *Size* : small quorum size implies less message sendings, thus reducing the communication cost of the system.
- *Load* : given a probability distribution on quorums access, this parameter measures the load of elements in handling requests. If the load is low, then each element is less required, thus it is free to perform other unrelated tasks.
- *Availability* : given that elements fail according to some probability distribution, this parameter indicates the probability that a quorum system is available, i.e. there is a quorum that consists of non-failed elements.
- *Cost of failures* : it measures the overhead message complexity due to failures. Informally,

the cost of failures is the additional number of elements that need be contacted when a failure occurs.

In this work, as we introduce the self-stabilization property, the performance of our quorum systems is also evaluated in terms of a new parameter - *the self-stabilization time T*. *T* can be considered as a measure of both *the availability* and *the cost of failures* criteria. Nevertheless the other criteria are briefly discussed later.

2.3 Self-stabilization Considerations

2.3.1 Self-stabilization Overview

The self-stabilization concept was introduced by Dijkstra [Dijk74] in the context of distributed systems. A self-stabilization system is a distributed system which can be started in any *possible* global state. Once started, the system spontaneously regains its consistency, without any sort of outside intervention. The self - stabilization property is very useful for systems in which processors may crash and recover spontaneously in an arbitrary state. When the intermediate period in between one recovery and the next crash is long enough, the system gets stabilized. More formally, we define self-stabilization for a system S with respect to a predicate P over its set of global states where P is meant to identify its correct execution. S is self-stabilizing with respect to P, if it satisfies these two properties [GE90], [AG93]: (i) *Closure*: P is closed under the execution of S, i.e. once P is established in S, it may not be falsified; (ii) *Convergence*: starting from any global state, S is guaranteed to reach a global state satisfying P within a finite number of state transitions [Sch93], [Dijk74]. The states satisfying P are called *legitimate* states. These two strong properties are used to theoretically demonstrate the ability of self -stabilizing systems to tolerate *transient failures* and dynamic topologies.

We introduce the model of failure that self-stabilization can tolerate. A transient failure is an event that may change the state of a system, but not its behavior. We assume that the state of a system is violable, whereas its behavior is not. Transient failures may change the global state of a system by corrupting the local state of a process as represented by memory or program counter or by corrupting message channels or shared memory. The self-stabilization property models the ability of a system to recover from transient failures on the assumption that they do not continuously occur.

2.3.2 Model of the Self-stabilizing System

We model our distributed information system S by its communication graph $G = (V,E)$ where nodes $i \in V$ are sites P_i and edges (i,j) are undirected communication links between P_i and P_j. Because symmetry is one of the factors that prevent self-stabilization, we suppose that P_0 is a *special site*, and all the other sites are identical. When it is necessary, a self-stabilizing leader election may be launched [DIM97b].

As the paradigm of self-stabilization is very general and does not depend on the communication media used by the system's sites, we have adopted the *link register model*~[DIM90] in which communication between two adjacent sites P_i and P_j is achieved by reading and writing two shared registers r_{ij} and r_{ji}, associated with the link between P_i and P_j. Site P_i can read from register r_{ji} of any adjacent site P_j and write to its own registers r_{ij}. Roughly, in such a model one can imagine the output message buffers as being sites link registers. The link register model is a variant of the shared memory model that is the nearest one to the reality where communication is based on message passing. In [DIM97a] Dolev, Israeli and Moran present a general scheme for simulating any self-stabilizing shared memory protocol by a self-stabilizing message driven protocol. They implement read/write operations by using a self-stabilizing token passing protocol, and certify the correctness of simulation.

Finally, we assume that sites are activated by a demon[1] (scheduler) such that within each

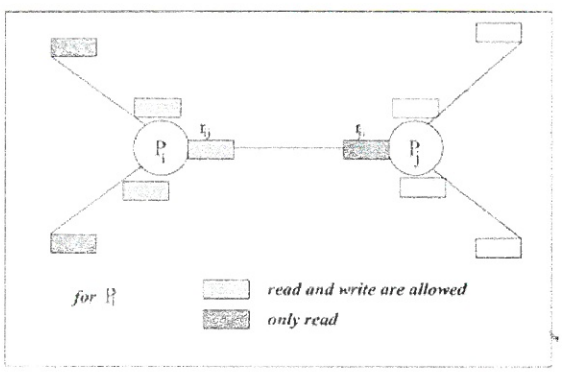


Figure 1

[1]for more detailed aspects on demon one can refer to [Dol95]

execution, each site occurs infinitely often. Our failure model considers both the site and the link failures. Sites fail by either stopping or modifying temporarily the memory state, but they never modify their own code. When a site fails, the site and all incident links are ignored. Link failures can also be memory corruption or broken links.

3. Proposed Self-stabilizing Solutions

3.1 The First Self-stabilizing Strategy

3.1.1 Informal Algorithm Description

The basic idea for constructing a self-stabilizing quorum system consists in interleaving the hierarchical composition function of quorums to a self-stabilizing spanning tree. The spanning tree protocol is a distributed Breadth First Search (BFS) protocol rooted on a special site. The output spanning tree is a BFS tree of the system communication graph.

Let G(P,E) be a graph, and let P_0 be the root of the tree. All the other sites are identical [DIM90]. Each register is a record with four fields: (1) r_{ij}.*father* is a binary field equal to one (resp. zero) if P_j is (resp. not) P_i's father in the spanning tree, (2) r_{ij}.*distance* is an integer that represents distance of P_i from root P_0, (3) r_{ij}.*coterie* is the local coterie calculated in P_i, corresponding to the coterie of a subtree rooted on P_i and (4) r_{ij}.*cotracine* stores global coterie corresponding to the spanning tree rooted on P_0. The *cotracine* field computed on the root is propagated by copy all over the system and stabilized later on. When a site executes its code, it copies into its own registers the value of the *cotracine* field from its father's register. We can distinguish two behaviors : one for the root and one for the other sites. Generally, the task of a site P_i consists in:

- reading its neighbors registers,
- looking for its father: the neighbor with the minimal distance from the root,
- identifying its children among its other neighbors,
- performing some local computation concerning quorums according to its position on the tree,
- updating its own registers.

The root will not execute the second and the third tasks: the root does not have a father node and all its neighbors are children. The content of the fourth field, which is calculated on the root site, will be propagated and stabilized by copy all over the system, after a finite number of steps.

Our algorithm uses a function called *Compute-quorums* that computes, using local information about the current spanning tree, a quorum system.

Compute-quorums defines a quorum system over a tree. A coterie is clearly defined when the tree depth is equal to 1, otherwise quorums can be calculated using an arborescent composition function [Ray92]

When the tree depth equals 1 (a_1: root ; a_2, a_2, ..., a_n,: leaves), quorums are given by :Q = $\{\{a_1,a_2\}, \{a_1,a_3\}\ldots, \{a_2,a_3,\ldots,a_n\}\}$.

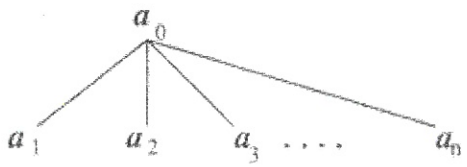


Figure 2. For A Tree with Depth 1

Arborescent Composition Function

Let T_x be the hierarchical composition function that allows quorums composition over the largest set of elements. We consider two sets of quorums Q_1 and Q_2 defined over sets U_1 and U_2 respectively, with $U_1 \cap U_2 = \phi$ and $x \in U_1$. To construct a quorum system over $U_3 = (U_1 \backslash \{x\}) \cup U_2$, we define T_x as follows:

$$Q_3 = T_x(Q_1,Q_2) = \left\{ q_3 \,/\, \forall q_1 \in Q_1, q_2 \in Q_2 : q_3 = \begin{cases} (q_1 - \{x\}) \cup q_2 & \text{if } x \in q_1 \\ q_1 & \text{if } x \notin q_1 \end{cases} \right\}$$

This composition preserves all quorums properties. Q_1 and Q_2 can be sets of elementary quorums, or quorums resulting from previous compositions.

As constructing quorums is a self-stabilizing spanning tree function, the resulting quorum system is self-stabilizing too.

3.1.2 The Basic Pseudo-code

The special site (the root) reads its neighbor registers, computes quorums and updates its own registers.

The root

```
DO forever
    for all neighbors m do      \* reading neighbors registers *\
        ir_mi ← read(r_mi);
    Q ← {};
    Q ← compute-quorum;
    for all neighbors m do      \* updating registers *\
        r_im ← write ⟨0, 0, Q, Q⟩;
OD
```

Each site other than the root looks for its father and children before calculating quorums and updating its registers.

Another site P_i

```
DO forever
    for all neighbors m do      \* reading neighbors registers *\
        ir_mi ← read(r_mi);
    Q ← {};        \* initialization of local variables *\
    first-found ← false;
    dist ← min (ir_mi.distance) + 1;
    for all neighbors m do      \* looking for father *\
        if (not first-found) and (ir_mi.distance = dist - 1)
        then
            father ← m;
            first-found ← true;
    Q ← compute-quorum \* calculation of the quorum system *\
    for all neighbors m do      \* updating registers *\
        if (m = father) then
            r_im ← write ⟨1, dist, P, ir_fatheri.pathroot⟩;
        else
            r_im ← write ⟨0, dist, P, ir_fatheri.pathroot⟩;
OD
```

Fonction *compute-quorum*(i)

```
leaf-child = notleaf-child = fictitious-child ← {};
for all neighbors m do
    if (ir_mi is a leaf ) then
        leaf-child ← leaf-child ∪ {m};
    else
        notleaf-child ← notleaf-child ∪ {m};

for each j in notleaf-child                    \* fix a correspondent *\
    fictitious-child ← fictitious-child ∪ {l};     \* fictitious child l *\

Q ← simple-coterie(i, fictitious-child ∪ leaf-child);
for each l in fictious-child
    Q ← composeT_l(Q, ir_jl.coterie);
return(Q);
```

3.2 A Self-stabilizing Planar Quorum System

We are improving our previous results and propose a new construction of self-stabilizing quorums based on a particular planar graph [Baz96]: the triangle lattice. The basic idea consists in using a self- stabilizing spanning tree in order to dynamically organize the sites of the network into a triangle lattice, giving to each site a unique number, and thus the associated quorums. Besides the planar quorums properties, those of having small size, optimal load and high availability, the advantage of our approach lies in that this new construction of quorums spontaneously tolerates transient faults and adapts to the dynamic configuration of the network, because of its self -stabilizing properties. The performance analysis based on simulations shows that the self -stabilization time of this new construction of quorums is better than that of our previous algorithm.

3.2.1 A Planar Quorum System Based on the Triangle Lattice

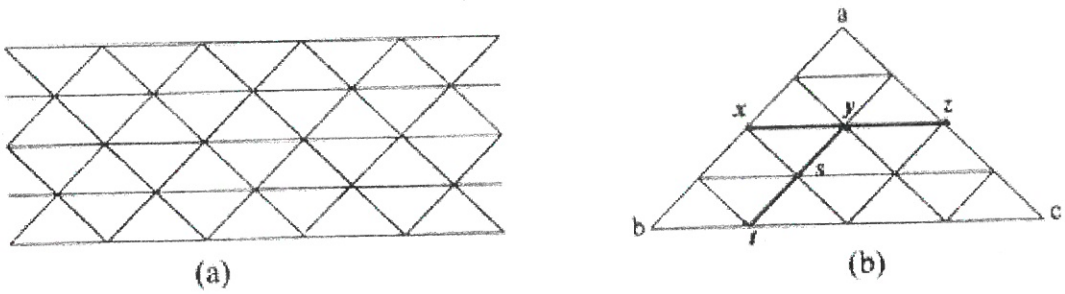
Recently, Bazzi introduced in [Baz96] a new class of quorum systems based on a particular planar graph : the triangle lattice. In [Baz96], it has been shown that quorum systems built over such a structure have small size, optimal load, and high availability.

The triangle lattice is a particular instance of a planar graphs class. For each integer d, we define a triangle lattice consisting of $\frac{d^2+d}{2}$ vertices connected as shown in Figure 3. We first define the infinite triangular lattice : it is the infinite graph whose vertices are points from (i,j) or $\left(i+\frac{1}{2}, j+\frac{1}{2}\right)$ where i and j are integers such that each vertex (x, y) of the graph has the vertices $(x+1,y)$, $(x-1,y)$, $\left(x+\frac{1}{2}, y+\frac{1}{2}\right)$, $\left(x-\frac{1}{2}, y+\frac{1}{2}\right)$, $\left(x+\frac{1}{2}, y-\frac{1}{2}\right)$ and $\left(x-\frac{1}{2}, y-\frac{1}{2}\right)$ as neighbors. The triangle lattice is a finite subgraph of the infinite triangular lattice whose vertices (i,j) satisfy $0 \le j \le \frac{d-1}{2}$ and $j \le i \le d-1-j$ for some positive integer d. An optimal quorums selection strategy was proposed in [Baz96].

Let G be a connected planar graph organized as a triangle lattice. In such a structure, we distinguish among three vertices a, b and c as shown in Figure 3 (b). A planar quorum consists of a minimal path P_h (horizontal) that connects (a,..., b) and (a,..., c), and a minimal path P_v (vertical) that connects P_h and (b, ..., c). *The resulting quorum systems are coteries and are non-dominated* [Baz96].

3.2.2 Proposed Self-stabilizing Strategy

In this Section a strategy for constructing self-stabilizing planar quorums over a general graph, is presented. We do not make any assumption on the system topology. The principle of our approach is based on a logical organization of the network into a triangle lattice T. According to our model, the vertices of graph T correspond to the sites in the system while the edges of T are logical edges which must not necessarily correspond to actual communication links between sites. Then the planar quorums can be automatically deduced from the logical structure.



(a) (b)

Figure 3. (a) Infinite Triangle Lattice ; (b) Triangle Lattice; *d=5*

Principle

This Section deals with the problem of constructing planar quorums without any prior knowledge of the underlying communication network. In a triangle lattice, a site can identify the position of its neighbors with the help of their coordinates as defined above. In our case we use site numbers rather than coordinates. The basic idea of constructing planar quorums over a general graph is to logically organize the graph as a triangle lattice, using a spanning tree. This spanning tree allows the numbering of all sites, which defines a unique order over the sites of the network. A logical triangle lattice is then built over the system using this order. The position of a site, level and rank in the triangle can be easily deduced from the number that represents it. Each site can know the numbers of its neighbors. There is a geometrical progression that defines an induction function between the site numbers.

Example

Consider G as a distributed system with ten sites P_0, ..., ,P_9 as depicted in Figure 4 (a). Let P_0 be a special site. A spanning tree of the network of which the construction is described in the next Section is given in Figure 4 (b). The generated paths are: $\{P_0, P_1, P_4, P_8\}$, $\{P_0, P_1, P_5\}$, $\{P_0, P_2\}$, $\{P_0, P_3, P_6, P_9\}$, $\{P_0, P_3, P_7\}$ to which corresponds the following order : P_0, P_1, P_4, P_8, P_5, P_2, P_3, P_6, P_9, P_7.

$P_9, P_7\}$, $\{P_0, P_1, P_8, P_3\}$ will define correct planar quorums.

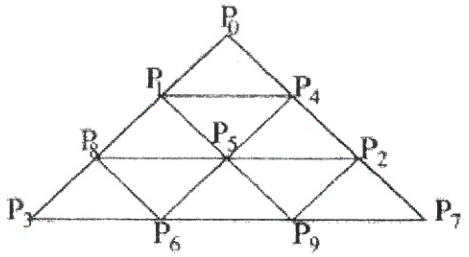


Figure 5. The Corresponding Triangle Lattice

Formally, let N_i be the site number of a site P_i. The P_i position in the triangle lattice can be defined by determining both its level d and rank r which can be calculated as follows:

$$d = k + 1 \left\{ \frac{k^2 + k}{2} < N_i \right\}$$

$$r = \frac{d - d^2}{2} + N_i \quad k = \left[\sqrt{2 * N_i}\right]$$

As mentioned above, a quorum q can immediately be determined. It consists of a union of a vertical path P_h and a horizontal one P_v given by the following expressions:

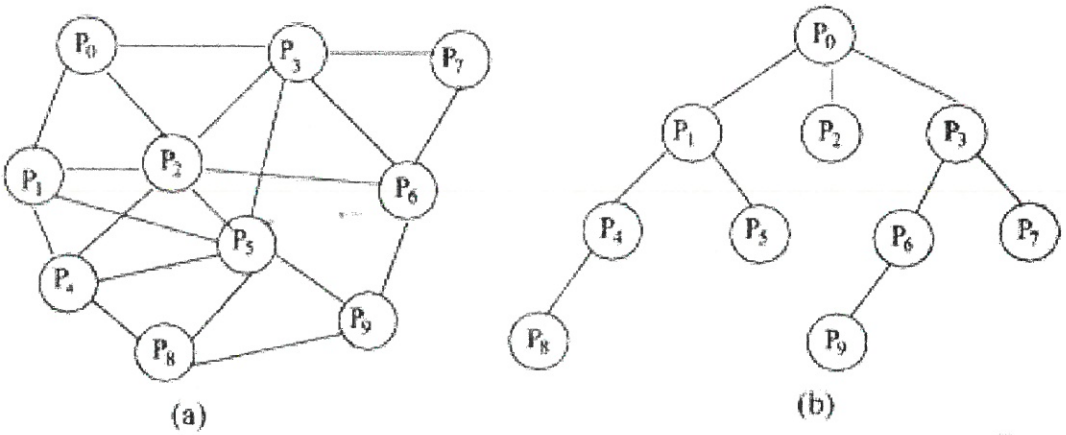


Figure 4. (a) The Communication Graph ; (b) BFS Spanning Tree

According to this order the sites can be organized into the following triangle lattice: sets $\{P_1, P_4, P_5, P_6\}$, $\{P_8, P_5, P_2, P_9\}$, $\{P_3, P_6,$

$$P_h = \left\{ N_i - (r-1), N_i - (r-2), \ldots, N_i, \ldots, \frac{d^2 + d}{2} \right\}$$

$$P_v = U_{n=0} \left\{ u_n^i / u_n^i < N \right\}$$

where N is the network size supposed to be known and u_n^i a series whose general term is given by :

$$u_n^i = N_i$$

$$u_n^i = u_{n-1^i} + d + n$$

Proposition 1 *For each* $N_i : q_i = P_h \cup P_v$ *defines a planar quorum.*

Hence, according to the previous proposition, one can easily build a planar quorum system using a spanning tree. In order to compute an adaptive planar quorum fitting the dynamic evolution of a communication network, our basic idea consists in building a self -stabilizing spanning tree over S, which allows to generate the same order stored in **each node** ; *sites are numbered according to the order in which they appear on the paths from root to leaves.* Then we obtain a logical triangle lattice. Planar quorums follow by selecting any two sets of nodes of which union defines horizontal and vertical paths in the triangle. When sites or links fail, the self -stabilizing spanning tree is spontaneously reconstructed and eventually leads to a new stabilized spanning tree which generates the corresponding planar quorums.

We adopt the same communication model. Each register consists of four fields: (1) r_{ij}. father is a binary field that is equal to 1 (resp. 0) if P_j is (resp. not) P_i's father in the spanning tree, (2) r_{ij}. *distance* is an integer that represents distance of P_i from root P_0, (3) r_{ij}. *path*: it corresponds to a set of all possible paths from P_i to descending leaf sites on the tree, and (4) r_{ij}. *pathroot* stores all possible paths on the tree that start from the tree root P_0 and end on a leaf site. Each site knows the site numbering once it correctly gets the *pathroot* field.

The Algorithm

Thereafter follow the pseudo-codes of the root site and of any other site. The special site (the root) reads its neighbors registers and generates all possible paths from the root to leaf processes.

4. Correctness Proof

Since the two constructions are based on a self-stabilizing spanning tree, the demonstrations of the self-stabilization property of the resulting quorum systems are similar. In this Section, we give the correctness proof of the second algorithm. Recall that the basic idea is to construct a self-stabilizing spanning tree from which we derive some order between the sites. Finally, a self-stabilizing planar quorum system can easily be deduced.

To prove the self-stabilization property of quorums, we first define the predicate that specifies the legitimate states and then prove the closure and convergence properties.

Predicate: Let P be the following predicate,

1. $\forall\ i \neq 0, \exists!\ j$ such that :

$$\begin{cases} r_{ij}.\ father = 1 \\ \forall\ \ k \neq j\ \ (i,k) \in E, \quad r_{ik}.\ father = 0 \end{cases}$$

2. $\forall\ \ (i,j) \in E : r_{ij}.\ distance = \min_k d_{oi}^k$

3. $\forall\ \ (i,j) \in E :$

$$\begin{cases} r_{ij}.\ path = \{ l_{ik} / P_k \ leaf\ process\ descending\ of\ P_i \} \\ r_{ij}.\ pathroot = \{ l_{0k} / P_k \ leaf\ process\ descending\ of\ P_0 \} \end{cases}$$

4. $\forall$ j such that $(0, j) \in E$ we have:

$$\begin{cases} r_{0j}.\ father = 0 \\ r_{0j}.\ distance = 0 \end{cases}$$

(1) guarantees father unicity for each process;

(2) guarantees that each P_i is at minimal distance from the root;

(3) guarantees numbering unicity;

(4) is a particular root property.

The root

```
DO forever
    for all neighbors m do      \* reading neighbors registers *\
        ir_mi ← read(r_mi);
    P ← {};
    for all neighbors m do      \* inserting the root identity   *\
        for each q in ir_mi.path do
            P ← P ∪ (P0⊕ q);
    for all neighbors m do      \* updating registers    *\
        r_im ← write ⟨0, 0, P, P⟩;
OD
```

For a site other than the root site, first it looks for its father and children, then generates descending paths and updates its registers.

Another process P_i

```
DO forever
    for all neighbors m do      \* reading neighbors registers *\
        ir_mi ← read(r_mi);
    P ← {};     \* initialization of local variables *\
    first-found ← false;
    dist ← min (ir_mi.distance) + 1;
    for all neighbors m do      \* looking for father    *\
        if (not first-found) and (ir_mi.distance = dist - 1)
        then
            father ← m;
            first-found ← true;
        else
            if (ir_mi.father = 1) then
                for each q in ir_mi.path do     \* inserting the identity    *\
                    P ← P ∪ (Pi⊕ q);
    for all neighbors m do      \* up-dating registers *\
        if (m = father) then
            r_im ← write ⟨1, dist, P, ir_fatheri.pathroot⟩;
        else
            r_im ← write ⟨0, dist, P, ir_fatheri.pathroot⟩;
  OD
```

The *closure* property is trivial. *Convergence* property can be decomposed into two steps. We introduce a first lemma to show that the spanning tree will stabilize in a finite number of steps, and a second one to prove that all generated orders converge to the same order. Finally we deduce from the two previous lemmas, the self -stabilization characteristic of our planar quorum system.

Lemma 1 *The spanning tree is built in a finite number of steps and stabilizes starting from any initial state.*

Proof: The first lemma can be shown by induction on the root excentricity that is defined as : e_0 = {max d_{0j} , 1≤ j≤ n} (d_{0j} : being the shortest distance of P_j from root P_0).

Let d be an integer and H the *induction assumption* defined as follows: **H** : ∀ $P_i \neq P_0$ where $d_{0i} \leq d$, P_i converges to a state where :

(1) P_i has a unique father, and

(2) all *distance* fields store the minimal distance of P_i from P_0, that equals the father's distance plus one.

In the case of d=0, the graph is reduced to a single node. H is obviously satisfied. Assuming that H is true for a tree with a root excentricity $e_0 = d$, we show that H is true for d+1.

Induction steps : considering a tree with a root excentricity $e_0 = d+1$, we assume that P_0 is the root tree. Given an infinite execution starting from an arbitrary configuration. According to the induction assumption, any process at distance ≤d converges to a state where (1) and (2) are satisfied. So, the subgraph formed by the processes that are at distance ≤d from P_0, stabilizes after a finite time. Since any execution can modify process states after stabilization, and because there is at least one P_i such that its real minimal distance from P_0 is d+1, once executed its code, it increments its father's distance by one and thus reaches a legitimate state where its distance is d+1. Therefore, a spanning tree of depth d+1 is built over G. According to the closure property, the tree structure cannot be modified. □

The generated orders are improved while the spanning tree is being constructed. However, we consider the worst case in Lemma 4.

Lemma 2 : *From a stabilized tree and an arbitrary memory state , all generated orders converge to a same one.*

Proof : We assume that a tree rooted on P_0 is built over graph G. This means that properties on the father and distance fields are satisfied. As already seen, each P_i must generate some local paths stored in the fields r_{ij}. *path* of its registers. The construction of local paths is in fact a restriction of the problem over a subgraph, which a spanning subtree rooted on P_i corresponds to. If we start from a state with arbitrary order, a process that executes its code can easily know its position on the tree: its father, its children and its other neighbors. Constructing local paths is to insert process identity into all children paths. Thus, for a leaf process local paths are restricted to its identity. A process on level 1 of the tree has a set of paths in the form (its own identity, the child identity) and so on until the root. All possible paths on the tree are generated in it and propagated by copy all over the system. Constructing paths is a tree function, and since the spanning tree is self -stabilizing, these paths will stabilize in a finite number of transitions.

Since each process activation implies the destruction of illegal paths, the system will reach in a finite time a global state in which all generated paths are legal and correspond to the spanning tree. Hence, the algorithm generates some order that defines a sites self-stabilizing numbering and consequently a self -stabilizing triangle lattice. □

Proposition 2 *A quorum system built over a self-stabilizing triangle lattice is self-stabilizing itself.*

Our quorum systems are first of all fault-tolerant. They offer full and automatic protection against all transient failures because of the self-stabilizing algorithm which can recover from any arbitrary configuration, no matter how much the data have been corrupted by failures. And they are also dynamic, because computing quorums represents a topology dependent function. After the occurrence of a topological change, the system runs for a while until its topology stabilizes and then converges to a new solution.

5. Performance Analysis of Quorum Systems

Constructing quorums is a spanning tree dependent function which is self -stabilizing. So, for performance analysis purpose, we focus on the self-stabilizing time. This Section makes an estimation of the number of steps to be taken so that the quorum system should stabilize. It is a difficult task to do not only because of the fact that it depends on the graph characteristics and the assumptions made on the algorithm execution, but also because there is no detection of stabilization, i.e. it is not possible to observe from within the system that a legitimate state has been reached. It is also difficult to know how long it will take for the system stabilization.

Nevertheless, on certain assumptions, we can have an idea of estimating the number of steps necessary to converge to a legitimate state. In the beginning a naive computation of the stabilization time is made. Then in order to validate our theoretical predictions, we make use of a simulation that estimates the self-stabilization time for different network topologies.

Finally, we model the stabilization problem with the help of the Markov chains theory. On the one hand, we describe how the

characteristics of a system with the best stabilization time can be defined. On the other hand we explain how to get an idea of the capacity of a given system in terms of time taken to stabilize. The idea of these empirical measures is a local checker that validates the stability situation [BAV91], [Var92]. Thus, if the protocol is in an inconsistent state, some components of the system will be able to notice this fact locally. We describe the legitimate state of the system in terms of local predicates. In a shared memory model, a local predicate is any predicate that only refers to the state variables of a pair of neighbors. If all local predicates hold, the system is stabilized. Recall that the protocols that are locally checkable but work on a tree topology, can be stabilized in time proportionally to the height of the tree [Var92],[GVG95].

5.1 Standard Computation of Self-stabilization Time

The number of steps the quorum system needs to stabilize depends on the graph characteristics and the demon under which the system works. The evaluation of this number will be restricted when a sequential demon is used. In a general case the situation is more complicated, for example when using any centralized demon, the sites can be activated in an arbitrary order. Nevertheless, with a distributed demon the results are better as many sites can be simultaneously activated.

Let e be the root site excentricity which gives the distance of the longest path from root e = Max $\{d_{0i}\ 0 \leq i \leq N\}$ (d_{0i} is the minimal distance of P_i from the root), and N the network size giving the number of sites. We define a sequential demon as a centralized demon in which the number of steps between two activations of a site is at most 2*N.

For the two constructions which are based on a spanning tree, the sites behavior will be the same. The difference between our two algorithms lies in the quorums computing function. The algorithms have the same stabilization time in terms of the number of steps. For example, the lower bound of the stabilization time of the second construction can be evaluated when the best sequential demon is supposed, reducing the self - stabilization time as much as possible. In such a case, the leaf sites first get their correct partial paths of the spanning tree. With another wave, these correct paths are propagated to their father sites, and so on, until the root site. It takes $e*N$ steps. Once the root site obtains paths from its children, these partial paths are completed and propagated to its sons, then its grandsons, and so on, until the leaf sites. Thus the complete propagation takes other N steps. The total number of steps is then $(e+1)*N$ under the best sequential demon.

Under a *general* sequential demon, the number of steps is $O(N^2)$. Indeed the time necessary to reach a legitimate state includes a time $O(N*e)$ for constructing the spanning tree plus a time $O(N)$ for computing and propagating the paths. Given $e \leq N-1$, the total number of steps necessary for stabilization is $O(N^2)$.

Proposition 3 *Using a sequential demon, the number of steps necessary for the stabilization of quorums is $O(N^2)$.*

To validate our predictions, we make some simulation experimentations that estimate the self-stabilization time for different network topologies.

5.2 Experimental Results

To validate our quorums constructing technique and to compare our theoretical predictions, we have carried out some simulation experiments. The tests for self-stabilization are done for some classical topologies when a general demon is used, and are compared with the stabilization time of the best sequential demon. The simulation results show how the number of steps required for the stabilization varies according to the network size.

Graphs 6.1, 6.2, and 6.3 show the evolution curves of the number of steps necessary for the stabilization of our algorithm according to the network size. Each graph contains two curves. The upper curve gives the stabilization time for some classical topologies under a general sequential demon, while the lower one exhibits the same stabilization time when the best sequential demon is used, giving the lower bound of the stabilization time.

In Graph 6.3, we can observe that the upper and the lower curves coincide for a star. That is due to the fact that all sites but the root site have the same characteristics, and therefore the stabilization time does not depend on the demon. The number of steps is just a linear function of N, as e is constant.

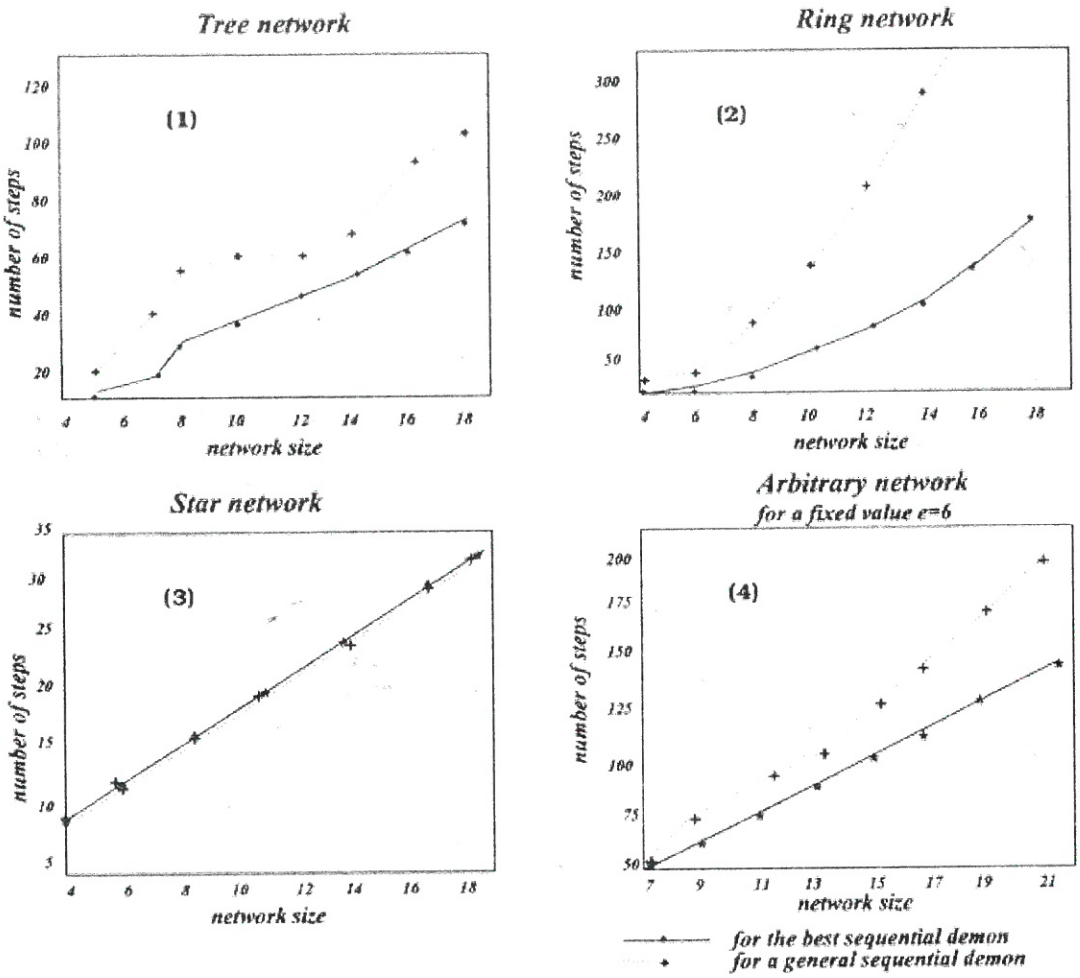


Figure 6. Results of Simulation

For a ring, the excentricity, which is [N/2], linearly increases with the network size, the stabilization of the algorithm takes about N*e, i.e. $N^2/2$ time as confirmed by Graph 6.2.

Graph 6.4 illustrates the number of steps required for stabilization when a general topology is used. We have kept the special site excentricity as constant e=6. We can see that the lower bound shown by the lower curve is proportional to the network size. The upper curve will exhibit results when an arbitrary sequential demon is used. We can see that the number of steps the system takes to stabilize is approximatively linear with the network size.

Based on the stabilization time, which is an important parameter in the design of dynamic self-stabilizing systems [BBBC97], we have also compared the two self-stabilizing quorum systems in terms of quorum size, load and availability.

Considering the simulation results, this comparison shows that the second strategy is more interesting in terms of stabilization time. As planar quorum systems are immediately deduced from the logical structure built over the system (sites do not perform any quorum computing), this strategy considerably improves the self -stabilization time. However one may overlook this advantage if the comparison is based only on the evolution of the number of steps required for stabilization.

On the other hand, according to the criteria of quorum size, load and availability of sites, the so constructed quorum systems are much better, because they are non -dominated, and all quorums have the same size $\cong \sqrt{2 * N}$, where N is the network size. This is a desirable property that provides a balanced system. Quorum systems are available beyond the stabilization period.

5.3 Modeling Using Markov Chains

To evaluate the reliability and the performance of our system we use the Markov chain concept. For this we propose a stochastic model [Bui92], which describes the evolution of the system through a state-machine approach.

This type of modeling let us study the system behavior, and, subsequently, deduce some optimality criteria. First it consists of selling a network model as homogeneous and finite as Markov chains in a steady state functioning, and of defining, by an analytical resolution, an optimal functioning rule that guarantees the best stabilization time. Finally, the estimation of probability terms of the transition matrix can be done using statistical results of simulation.

A Markov chain is a discrete-time stochastic process defined over a set of states S in terms of a matrix P of transition probabilities. Set S is either finite or countably infinite. The transition probability matrix P has one row and one column for each state. The Markov chain is in one state at any time, making state-transitions at discrete time-steps $t = 1, 2, \ldots$. Entry M_{ij} in the transition probability matrix is the probability that the next state is j, given the current state i. An important property of a Markov chain is the *memorylessness property*. The future behavior of a Markov chain depends only on its current state, and not on the way it took to get to the present state. More formally, this property can be stated as follows:

$$\forall t, P\{X_{t+1} = j / X_0 = i_0, X_1 = i_1, \ldots, X_t = i\} = P\{X_{t+1} = j / X_t = i\}$$

X_t is the state of the Markov chain at time t.

5.3.1 Proposed Stochastic Model

Our stochastic model based on the approach in [Bui92] is represented by a network of N processes P_0, P_1,...,P_{N-1}. Each one defines a homogeneous and finite Markov chain. The behavior of a process P_k is a random variable which satisfies the memorylessness property X^k which is finite and time-nondependent. The associated Markov chains are ergodic and without cycle. We obtain a network of homogeneous and finite Markov chains X^0, X^1, ..., X^N in a steady -state functioning. The parameters of such X^k are :

State space : Σ consists of three states. A process P_k can be in either

1. *stable state* : the system is locally stabilized ; quorums are correct and available. We use a local checker that validates the stability situation, or

2. *not-stable state* : the process behavior has not been stabilized yet and the quorums computation is in progress. Quorum systems are considered as non available, or

3. *out-of-service state* : the corresponding site is down (fail stopping) which induces a change in the current topology.

Transition matrix: M^k depends on a parameter $\rho_k = (\alpha_k, \beta_k, \phi_k \gamma_k,) \in [0,1]^4$. The corresponding state automaton is given by :

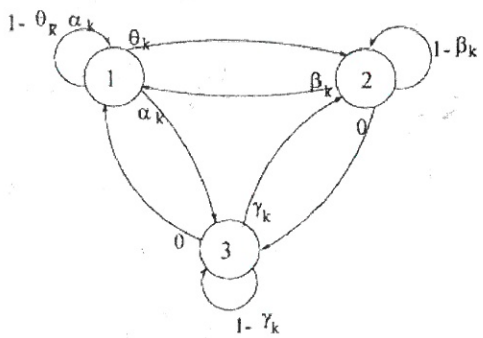


Figure 7. The State Automaton for P_k

So, at any moment, X^k is susceptible of being in one of the three states (1) *stable*, (2) *not-stable* and (3) *down*. These states are all recurrent and have the same period equal to 1. There is one ergodic class without cycle, so the system is in a steady -state functioning.

For the sake of clarity, this evaluation takes into account the following two assumptions, otherwise it induces a large state space: (1) We only consider the physical site failure, any other fault can occur, (2) site failures are not frequent, and a process P_k can never fail when it is in *nstable* state. Thus, the transition matrix M^k has the following form :

$$M^k = \begin{pmatrix} 1-\alpha_k-\theta_k & \theta_k & \alpha_k \\ \beta_k & 1-\beta_k & 0 \\ 0 & \gamma_k & 1-\gamma_k \end{pmatrix}$$

Initial law : X_0^k, which is not indispensable, we nevertheless suppose that initially all processes are in **nstable** state.

5.3.2 Optimization Problem

An analytical resolution of the problem consists in expressing it as an optimization problem. This allows the determination of a theoretical rule which will guarantee the best operation of the system, and a design of which implementation draws near this rule. The problem of the "good" functioning system is based on some optimality criteria, which are represented by optimization (maximization or minimization) of some functions "guiding" to the system functioning. We define such a function F called *choice function* of parameter $\rho = (\rho_1, \rho_2, \ldots, \rho_N)$ where $\rho_k = (\alpha_k, \beta_k, \theta_k, \gamma_k) \in [0,1]^4$, of which components are the transition probabilities corresponding to P_k.

This function optimization consists in finding the *optimal rules* M_ρ. These rules are called optimal if their ρ render the choice function F maximum (resp. minimum), when the optimality criterion is the maximization (resp. minimization) of F. A rule is called *bad* if its ρ makes the choice function F maximum (resp. minimum), when the optimality criterion is the minimization (resp. maximization) of F. A rule which is not bad is called *judicious*. Optimal or judicious rules of functioning are "good" functioning rules.

In this context, the optimization problem consists of minimizing the time a process takes to return to a *stable* state. A function F is designed, which must be optimized, for finding the optimal rule(s).

$$\min F = \min \sum_{k=1}^{N} T_1^k = \min \sum_{k=1}^{N} \sum_{n=1}^{\infty} f_{11}^{k(n)}$$

where $f_{11}^{k(n)}$ is the probability, which starting from a state 1 (stable), P_k returns to for the first time after n transitions, i.e:

$$f_{11}^{k(n)} = P\left\{ \begin{matrix} X_{t+n} = i, X_s \neq i, \\ s = t+1, \ldots, t+n-1 / X_t = i \end{matrix} \right\}$$

According to the above state automaton, the general form of $f_{11}^{k(n)}$ is given by :

$$\begin{cases} f_{11}^{k(1)} = 0 \\ f_{11}^{k(2)} = \theta_k \beta_k \\ f_{11}^{k(n)} = \theta_k (1-\beta_k)^{n-2} \beta_k + \alpha_k \beta_k \gamma_k \sum_{p=0}^{n-3} (1-\gamma_k)^p \left(1-\beta_k\right)^{n-3-p} \quad n \geq 3 \end{cases}$$

the goal is to find the optimal ρ_k that minimizes F

$\rho_{kopt} = (f_1(\theta), f_2(\theta), f_3(\theta), f_4(\theta))$ where $\theta \in [0,1]$.

By optimizing the stabilization time for each process, the global time of system stabilization will be reduced.

5.3.3 Estimations of A Site Behavior

In practice, the Markov chains are used as a tool for estimating some unknown parameters of the system concerned using the results of the Markov chains theory and some empirical measures. We have designed some procedure attempting to determine the characteristics of a given system and thus to give an idea of its capacity in terms of time it needs to stabilize.

To estimate the probability values of the transition matrix, we consider $\tilde{M}_k$, defined as a two-dimensional transition matrix associated with the randomized variable Y^k given by:

$Y_t(\omega) = (i, i') \Leftrightarrow X_{t-1}(\omega = i)$ and $X_t(\omega = i')$

Obviously, Y^k is also a Markov chain.

In studying the processes behavior from the pairs *(i,i')* we want to remark the number of transitions from one state to another and the length of time it goes from one state to another.

Thus we can use the following result, if the initial site is in a steady -state functioning, the two- dimensional one too and we can apply the "strong law of large numbers". We establish that:

$$\lim_{n \to +\infty} \frac{n_i}{n} = q_i = \frac{1}{T_i}$$

$$\lim_{n \to +\infty} \frac{n_{ij}}{n} = q_i m_{ij}$$

$$\lim_{n\to+\infty} \frac{n_{ij}}{n} = m_{ij}$$

where $n_{ij} = \sum_{t=1}^{n} 1_{\{(i,j)\}}\left[Y_t(\omega)\right]$ is the number of direct transitions from state i to state j, for the initial process (X_k) and $n_{i.} = \sum_{j=1}^{3} n_{ji}(\omega)$.

When the observance period n is long enough, we consider that $\frac{n_{ij}}{n_i}$ is a good approximation of m_{ij} and that $\frac{1}{q_1}$ is a good estimated value of T_1^k, the return time to the *stable* state for some process P_k. We can thus approximate the global time the system needs to stabilize. In the following an application of which results have an interesting interpretation in terms of the Markov chains theory, is presented.

Example

Let us consider as an example a small system with five sites P_0, P_1, P_2, P_3, P_4 connected as follows:

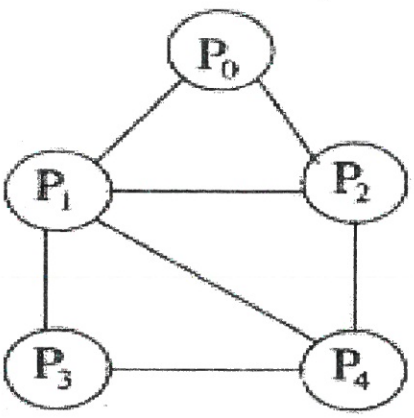


Figure 8. The Communication Graph

We use a simple probabilistic model of the failures and repairs in the system. For the sake of simplicity, we suppose that the root P_0 is a powerful (strong) site which is never down. All other sites may fail independently. We assume that failures and repairs follow a Poisson distribution with parameters λ_p and λ_r respectively. And we assume that failures are transient, that they are crash failures (i.e. a failed site stops function rather than functions incorrectly).

We use the *local detection* paradigm. The essence of this paradigm is in defining a local condition based on the state of a site and its immediate neighborhood, such that the system should be in a globally legal state if and only if the local condition is satisfied on all nodes.

We have chosen to study the behavior of a local site, as P_1 for example. In order to estimate the expected number of steps necessary for its stabilization, its evolution has been observed over a large time interval and the number of transitions between states has been counted. Using a sequential demon, the transition matrix M^1 estimated by :

$$\tilde{M}^1 = \begin{pmatrix} 0.9883 & 0.0104 & 0.0013 \\ 0.0541 & 0.9459 & 0 \\ 0 & 0.0099 & 0.9901 \end{pmatrix}$$

confirms the stationary distribution of X_1 given by:

$$\forall i,j \quad \lim_{n\to+\infty} m_{ij}^{(n)} = q_j > 0$$

We obtain

$$\lim_{n\to+\infty} \tilde{M}^{1(n)} = \begin{pmatrix} 0.7421 & 0.1605 & 0.0974 \\ 0.7421 & 0.1605 & 0.0974 \\ 0.7421 & 0.1605 & 0.0974 \end{pmatrix}$$

The expected return time to a *stable* state for site P_1 is given by $T_1 = \frac{1}{q_1}$. In this case our empirical measures estimate this parameter by (1.4). T_1 is the local stabilization time of P_1. Calculating this time for each site P_k under the same conditions, allows that the global time for the system stabilization is estimated. In this case, sites being activated in a sequential order, the global time corresponds to the sum of local stabilization times for all sites $T \cong \sum_k T_k$.

6. Concluding Remarks and Perspectives

Within the TransDoc project, we have proposed two self-stabilizing strategies for constructing quorum systems for reliable document access in

fully distributed information systems. Besides the good properties of planar quorums in terms of load and size, the advantage of our approach is in that the construction of quorums spontaneously recovers from transient faults and is self-adaptive to dynamic modifications of the network, thus providing high availability. Furthermore, both theoretical performance evaluation and experimentation show that our proposition is an efficient one.

6.1 Related Work

The work on quorum systems considers many questions about fault tolerance. Several authors [AA91], [Baz96], [RL92] have proposed different strategies of fault tolerance for constructing quorum systems. In planar quorum systems [Baz96], Bazzi introduced a failure procedure which tries to go around the failure region while constructing a horizontal path. Unfortunately this solution is not adaptive and cannot tolerate network changes there where sites are dynamically inserted. Another limitation of this strategy is when a failure occurs: both size of quorums and load are high. Considering the network organized into a tree, Agrawala and El Abbadi [AA91] constructed quorums by selecting paths from the root to leaves. When a failure occurs, they replace the failed site by a set of sites that belong to a path starting from this site and ending on a leaf. This proposition considerably increases the communication cost: a large number of messages and large quorum size. Furthermore their solution only tolerates a restricted number of failures and is not resilient to leaf failure. Generally, their algorithm suffers from poor failure resilience. Furthermore, the load of sites is not fairly balanced in the solution since the root site belongs to all quorums. Maekawa has proposed to construct "quorums" by imposing a logical structure on the network. In spite of the small quorum size, Maekawa's protocol only associates one quorum with each site. This makes the protocol be non -tolerant to failures and reduces the availability of the system. The construction used in [RL92] assumes some predefined order of sites to characterise a grid based quorum. In spite of its dynamic aspect this solution is not optimal in terms of quorums size.

6.2 Perspectives

We have applied our self- stabilizing quorum construction to resolve several concrete problems within the TransDoc project. The first one is a reliable document access control for ensuring the "one copy equivalence" property. Within the same project we have also studied an application on information retrieval in global information systems. In the following we set out the guidelines of these two applications.

1. A global information system such as World Wide Web (WWW) is a very dynamic system. It is difficult for the user to keep up with the fast pace of information generation. We have information providers that look for interested users and we have users that seek relevant information. The aim is to find a strategy to perform the matching and to establish the flow from providers to users. A quorum systems based solution is the selective dissemination of information servers (SDI) [YGM87]. A document (resp. user profile) must be sent to one or more "document quorum" (resp. "profile quorum") servers. The non-empty intersection of quorums guarantees that a profile misses no documents.

2. In an open information system, a service may be provided by more than one server. A client asks the system for a particular service by means of its name and not by its address because servers may be mobile. Thus before the client sends his request, he has to locate a server that provides the desired service. In a previous work we have proposed several multi-agent based dynamic request placement strategies [CRB97], [Ram97]. Actually, the mechanism that translates the name of a service into an address in the network can be dealed with quorums. Each server s posts at a set of nodes P(s) the address where it resides. This information is locally stored in each element of P(s). To request a service, the client selects a set Q(c) and queries each one of its elements. If P(s) and Q(c) are quorums, any element of $P(S) \cap Q(c)$ which is not empty, can return the address where the service is available.

REFERENCES

[AA91] AGRAWALA, D. and EL ABBADI, A., **An Efficient and Fault-Tolerant Solution for Distributed Mutual Exclusion**, ACM TRANSACTIONS ON COMPUTER SYSTEMS, 1991, Vol. 9, No. 1, February 1991, pp. 124-143.

[AG93] ARORA, A. and GOUDA, M.G., **Closure and Convergence: A Foundation of Fault Tolerant Computing,** .IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 1993, pp. 1015-1027.

[BAV91] SHAMIR, B.P., AWERBUCH,B. and VARGHESE, G., **Self-stabilization By Local Checking and Correction**, Proceedings of the 31st Annual IEEE Symposium on Foundations on Computer Science, FOCS'91, 1991, pp. 268-277.

[Baz96] BAZZI, R., **Planar Quorums**, Proceedings of International Workshop on Distributed Algorithms, WDAG, Bologna (Italy), SPRINGER- VERLAG, LNCS 1151, October 1996, pp. 251-268.

[BBBC97] BELKOUCH, F., BAÂLA, O., BUI M. and CHEN, L., **Self-Stabilizing Construction of Quorums**, Proceedings of European Research Seminar on Advances in Distributed Systems, ERSADS, Zinal Switzerland, March 1997, pp. 296-301.

[BC97] BELKOUCH, F. and CHEN, L., **Self-Stabilizing Planar Quorums**, Proceedings of International Conference on Principle of Distributed Systems, OPODIS Chantilly, France, December 1997, pp. 205-219.

[Bui92] BUI, M., **Tuning Distributed Control Algorithms for Optimal Functioning**, JOURNAL OF GLOBAL OPTIMIZATION, No. 2, 1992, pp. 177-199.

[CDF97] CHEN, L., DONSEZ, D. and FAUDEMAY, P., **Design of TransDoc: A Research Vehicle for Multimedia Documents on the Internet**, Proceedings of 3rd Basque International Workshop on Information Technology, BIWIT'97, IEEE COMPUTER SOCIETY PRESS, Biarritz, France, July 1997.

[Chen98] CHEN, L., **Une contribution pour les acces intelligents dans les systemes d'information globaux**. Rapport d'habilitation a diriger les recherches, Université de Technologie de Compiegne, January 1998.

[CRB97] CHEN, L., RAMOS, F. and BUI, M., **Stratégies de placement dynamique des requetes multi-agents pour les systemes d'information globaux**, CALCULATEURS PARALLELES, Vol. 9, No. 3, September 1997, pp. 347-366.

[Del95] DELAET, S., **Auto-stabilization: modele et applications a l'exclusion mutuelle**, Université Paris XI Orsay, 1995.

[Dijk74] DIJKSTRA, E.W., **Self-stabilizing Systems In Spite of Distributed Control**, ACM COMMUNICATIONS, Vol. 17, No. 11, 1974, pp. 643-644.

[DIM90] DOLEV, S., ISRAELI, A. and MORAN, S., **Self-stabilization of Dynamic Systems Assuming Only Read/Write Atomicity**, Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, Montreal, Canada, August 1990, pp.103-117.

[DIM97a] DOLEV, S., ISRAELI, A. and MORAN, S., **Resource Bounds for Self Stabilization Message Driven Protocols**, Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing, 1991, pp. 281-293, Journal version- SIAM ON COMPUTING, ACM, Vol. 26, No. 1, February 1997, pp. 273-290.

[DIM97b] DOLEV, S. ISRAELI, A. and MORAN, S., **Uniform Dynamic Self-stabilization Leader Election**, Proceedings of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Vol. 8, No. 4, April 1997, pp. 424-440.

[GE90] GOUDA, M.G. and EVANGELIST, M., **Convergence/Response Tradeoffs in Concurrent Systems**, Proceedings of the 2nd Symposium on Parallel and Distributed Processing, IEEE, December 1990, pp. 288-298.

[Gif79] GIFFORD, D.K., **Weighed Voting for Replicated Data**, Proceedings of 7th Symposium on Operating Systems Principles, December 1979, pp. 150-162.

[GVG95] ARORA, A., VARGHESE, G. and GOUDA, M.G., **Self-stabilization by Tree Correction**, Proceedings of the 2nd Workshop on Self-stabilizing Systems, 1995, pp. 12.1-12.14.

[Kum91] KUMAR, A., **Hierarchical Quorums Consensus: A New Algorithm for Managing Replicated Data,** IEEE TRANSACTIONS FOR COMPUTERS, 40(9), 1991, pp. 996-1004.

[Mae85] MAEKAWA , M., **A $\sqrt{n}$ Algorithm for Mutual Exclusion in Decentralized**

Systems, ACM TRANSACTIONS ON COMPUTER SYSTEMS, Vol. 3, No. 2, 1985, pp. 145-159.

[MB85] GARCIA MOLINA, H. and BARBARA, D., **How To Assign Votes in A Distributed System**, JOURNAL OF THE ACM, Vol. 32, No. 4, 1985, pp. 481-860.

[MV88] MILLENDER, S. J. and VITANYI, P.M.B., **Distributed Match-making**, ALGORITHMICA, Vol. 3, 1988, pp. 367-391.

[NW94] NAOR, M. and WOOL, A., **The Load, Capacity and Availability of Quorums Systems**, Proceedings of the 35th Symposium on Foundations of Computer Science, IEEE COMPUTER SOCIETY PRESS, November 1994, pp. 214-225.

[NW96] NAOR, M. and WOOL, A., **Access Control and Signatures Via Quorum Secret Sharing**, 3rd ACM Conference on Computer and Communications Security, March 1996, pp. 157-167.

[PW95] PELEG, D. and WOOL, A., **Crumbling Walls: A Class of High Availability of Quorum Systems**, Proceedings of the 40th Annual ACM Symposium on Principles of Distributed Computing, ACM, November 1995, pp. 120-129.

[Ram97] RAMOS, F., Placement dynamique des requetes multi-agents dans les systemes d'information globaux, These de Doctorat, June 1997.

[Ray86] RAYNAL, M., **Algorithms for Mutual Exclusion**, MIT PRESS, 1986.

[Ray92] RAYNAL, M., **Gestion des donnécs réparties: problemes et protocoles**, EYROLLES, 1992.

[RL92] RABINOVICH, M. and LAZOWSKA, E.D., **Improving Fault Tolerance and Supporting Partial Writes in Structured Coterie Protocols for Replicated Objects**, ACM, 1992.

[SBDS85] DAVIDSON, S.B., GARCIA-MOLINA, H. and SKEEN, D., **Consistency in Partitioned Network**, ACM COMPUTING SURVEYS, Vol. 17, No. 3, December 1985, pp. 341-370.

[Sch93] SCHNEIDER, M., **Self-stabilization**, ACM COMPUTING SURVEYS, Vol. 25, March 1993.

[Tho79] THOMAS, R.H., **A Majority Consensus Approach to Concurrency Control for Multiple Copy Database,** ACM TRANSACTIONS ON DATABASE SYSTEMS, Vol. 4, No. 2, 1979, pp. 180-209.

[Var92] VARGHESE, G., **Self-stabilization by Local Checking and Correction**, Ph.D Thesis, Massachusetts Institute of Technology, MIT/LCS/TR-583, MA, USA, 1992.

[YGM87] Yan, T.W. and Garcia-Molina, H., **Distributed Selective Dissemination of Information**, Parallel and Distributed Information Systems, IEEE Computer Society Press, September 1987, pp. 89-98.