Enhancing the Fault Tolerance of Replication: Another Exercise in Constrained Convergence

Felix C. Gärtner* and Henning Pagnia
Darmstadt University of Technology
Department of Computer Science
D-64283 Darmstadt, Germany
Email: {felix|pagnia}@informatik.tu-darmstadt.de

As computer systems influence an increasingly large part of everyday life, our society is becoming more and more dependent upon correct and well-designed systems. This is especially obvious in mission critical applications, like flight control systems, where it is important that the system continually provides the respective service even in the presence of internal faults. From the ability to hide the effects of these faults from the user, this approach has been called *masking fault tolerance* and has been the focus of research in fault tolerant computing for many years. However, it is a well known fact that in many scenarios masking fault tolerance is extremely expensive to achieve or even provably impossible [5]. In such cases, other notions of fault tolerance are desirable, which do not necessarily mask the effect of faults, but are still able to recover from them.

Along these lines, the concept of self-stabilization has emerged as a complementary paradigm to masking fault tolerance in distributed computing [4, 9]. A system is said to be self-stabilizing, if – starting from any state – it automatically recovers to a specified set of *legal* states in a finite period of time [9]. The arbitrary state from which the system starts resembles the effect of a transient fault within the system. Such a fault could be the corruption of local memory or the fact that messages have been induced into or removed from channels. During the recovery process the user might experience a limited amount of disrupted behaviour by the system, but guarantee is given that correct system operation will eventually resume. The fact that the type of fault is not specified further contains the striking power of the paradigm: The ability to mask the effect of faults is traded for the ability to tolerate *any kind and any number* of faults. Thus, self-stabilizing systems offer a degree of fault tolerance that goes beyond the shortcomings of masking fault tolerant systems. However, interest in this concept so far has been mostly theoretical.

In this abstract we plead for moving the benefits of self-stabilization into the application domain. We support our case by constructing an extendable self-stabilizing solution for an important problem in distributed computing: weak-consistency replication. Our approach adopts a design strategy known as *constraint satisfaction* [2, 3], which is a refinement of general algorithm composition. We will present this method in brief next and then apply it to the problem of guaranteeing mutual consistency in a network of replicas. As a conclusion we will shortly discuss the merits of our solution compared to existing schemes for the same problem. To ease presentation we will restrict ourselves to using the *serial model* of distributed computations [7]. A discussion of using the more realistic *distributed model* (which uses message passing) is contained in the conclusions.

Constraint Satisfaction. Starting point for every stability consideration is a specification of the set of legal states, usually given as a predicate P over the global system state. The idea of constraint satisfaction is to formulate P as a set of *constraints*. A constraint is a predicate on the local state of a node such that the conjunction of all constraints is equivalent to P. This ensures that overall states where $\neg P$ holds can be detected efficiently because in such cases at least one constraint c must be violated. The design methodology prescribes to design a *convergence action* for every constraint c_i at node i of the form:

$$\neg c_i \rightarrow \text{“establish } c_i\text{”}$$

We assume the serial execution model [7], in which nodes have access to the state of their neighbouring nodes. Here it is obvious that systems designed in this way do not necessarily converge to a state in P since correcting one constraint locally might invalidate another constraint at a neighbouring node. This point reveals the difficult part of the design method: convergence validation. Arora *et. al.*

*This author's work was supported by the German Research Association (DFG) as part of the “Graduiertenkolleg ISIA” at Darmstadt University of Technology.

[3] have published a first set of sufficient criteria that can be used to prove convergence. Their method is based on the notion of a *constraint graph* that – informally spoken – captures the update dependencies of the nodes' constraints. One result states, that convergence is guaranteed if the constraint graph is acyclic.

Simple Self-stabilizing Replication. We assume that each node i in the network has a local *replica* r_i and that the topology of the network is given by a *neighbouring relation* N. Due to the serial execution model, a node i can read the state of a node j iff (i, j) is in N. We say that the system is in a *mutually consistent state*, iff the following global predicate holds:

$$P \equiv \forall i, j : r_i = r_j \tag{1}$$

To bring P into a form which is more suitable for our purposes, we re-write Equation (1) as:

$$P \equiv \forall i, j : (i, j) \in N \Rightarrow r_i = r_j \tag{2}$$

If the network is connected and due to the transitivity of equality both forms are equivalent. Thus, P can be expressed as the conjunction of locally checkable predicates $r_i = r_j$ for all $(i, j) \in N$. So Equation (2) is suitable for applying constraint satisfaction.

We implement the algorithm by adding an action

$$r_i \neq r_j \rightarrow r_i := r_j$$

to the code of node i, one for every neighbour j. Now i depends upon j (meaning that a change of r_j implies a subsequent change of r_i). The next step is to remove actions from the code of the nodes in such a way, that the graph representation of the depends-upon relation is a tree containing all nodes. When this is done, the results of Arora *et. al.* [3] guarantee convergence to mutual consistency. As a result we obtain a simple scheme for self-stabilizing replication, which can tolerate any kind of transient faults.

Because of the non-masking nature of the paradigm, the consistency of data is necessarily weak (i.e. readers may get "old" values but only for a limited period of time). Due to our construction process the scheme follows the primary copy approach for replication with the primary copy being the root of the tree.

Conclusions. We have considered the problem of mutual consistency as an exercise in applying the method of constraint satisfaction. The result was a simple self-stabilizing algorithm for weak-consistency replication, which by nature can tolerate any kind of transient faults. This is a property which standard algorithms for the same problem [1, 6] do not have. In order to compare other properties of these solutions with ours, our scheme, which uses the serial execution model, must be transformed into the distributed model [7], that employs message passing. The transformation can be done by standard techniques [8] that preserve the fault tolerance properties. The result of the comparison shows that, while standard solutions provide masking fault tolerance, our scheme naturally stays non-masking.

The major contribution of our work however lies in the fact, that we started at a point where traditional masking fault tolerance considerations end. We have shown that the very desirable property of self-stabilization can be added rather mechanically through a well-defined construction process, which spares us from tedious correctness proofs. Due to the close relationship between group communication and replication [6] our approach can be extended to implement a self-stabilizing reliable broadcast. The next step is to evaluate the applicability of the stabilization approach in systems under the burdon of day-to-day usage.

References

[1] N. Adly and A. Kumar. HPP: A hierarchical propagation protocol for large scale replication in wide area networks. In *Proc. Int. Conf. on Computing and Information*, July 1995.

[2] A. Arora and M. Gouda. Load balancing: an excercise in constrained convergence. In *Proc. 9th International Workshop of Distributed Algorithms*, pages 183–197, 1995.

[3] A. Arora, M. Gouda, and G. Varghese. Constraint satisfaction as a basis for designing nonmasking fault-tolerance. In *Proc. Int. Conf. on Distr. Comp. Sys.*, pages 424–431, 1994.

[4] E.W. Dijkstra. Self stabilizing systems in spite of distributed control. *CACM*, 17(11):643–644, 1974.

[5] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[6] R. Golding. *Weak-consistency group communication and membership*. PhD thesis, University of California, Santa Cruz, December 1992.

[7] S.-T. Huang, L.-C. Wuu, and M.-S. Tsai. Distributed execution model for self-stabilizing systems. In *Proc. 14th Int. Conf. on Distr. Comp. Systems*, pages 432–439, 1994.

[8] M. Mizuno and H. Kakugawa. A timestamp based transformation of self-stabilizing programs for distributed computing environments. In *Proc. 10th Int. Workshop on Distr. Alg.*, pages 304–321, 1996.

[9] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.