Massachusetts Institute of Technology
6.895: Advanced Distributed Algorithms
Professor Nancy Lynch

Lecture 16
October 31, 2002

Self Stabilization (Part II)

Papers

- Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction. Based on FOCS 91 paper and Ph.D. thesis.
- Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. Self-stabilization by local checking and network reset. Proceedings of the 8th International Workshop on Distributed Algorithms, WDAG'94, 1994.

1 Common Background

1.1 Definition of Self Stabilization:

Despite a history of arbitrary corruption of data, once such faults stop, the system should stabilize to correct behavior without manual intervention.

- Copes with set of failures including link and node failures.
- No initialization required for self-stabilizing programs.

1.2 Criticism:

- Handles corruption of network state but not program code $\rightarrow$ Program code can have redundancy to take care of itself!
- Initial mistakes allowed $\rightarrow$ Leads to only temporary loss of service, so it is fine.
- Contrasted to Byzantine fault model $\rightarrow$ Orthogonal issues.

1.3 Previous Work:

Covered by Prof Patt-Shamir: Dijkstra, Token passing, Spanning tree and Network reset. Here we discuss efficient and general methodology for designing self-stabilizing protocols (Local Checking and Correction). Katz and Perry looked into centralized checking and correction. Now, the papers described have local decentralized checking. The first paper describes local checking and local correction. This is not suitable for all kinds of algorithms, hence the next paper describes Local Checking and Global Reset. This is a more powerful algorithm.

1.4 Local Checking and Correction: Preview

Consider a system where sender send at most Max messages at most at a time without acknowledgment and each time it gets a CR from the receiver it gets an extra credit C, for which it can send one more message M. The invariant in an acknowledgment based messaging scheme is that the the number of credits with sender (C), the number of messages in the channel(M), the number of messages queued up at the receiver(Q) and the number of acknowledgments(CR) add up to Max:

$$C + M + Q + CR = Max$$

Periodically sender stops sending messages and takes a snapshot of the receiver for the number of messages queued at the receiver. If

$$Q + C = Max$$

Then the system is in a correct state.

1.5 Extending to general network

The first paper (ASVD) proposes a framework for algorithms which work locally for checking and correction. With the help of such schemes, mutual exclusion, network resets and end-to-end communication across an unreliable network can be achieved.

1.6 Modeling Tools

Automaton components:

1. Set of Actions (A): input, output, internal
2. Finite set of states
3. A transition relation $R(A) \in$ states $\times$ actions $\times$ states

$$\forall s, a \exists (s, a, s\ ) \in R(A)$$

4. An equivalence relation part(A) partitioning the set of locally controlled actions into equivalence classes c such that

$$\forall c \in part(A), \exists t_c, t_c \geq 0$$

1.7 Terminology

Let a *timed element* (x,t) for an automata be a tuple with time t and x from an arbitrary domain like state or action. Then an execution α of automaton A is an alternating sequence of timed states and actions of A of form:

1. $s_0 \in start$ and $(s_i, a_{i+1}, s_{i+1}) \in R$ forall $i \geq 0$
2. The sequence can either be finite or infinite, but if finite it must end with a timed state.

3. The sequence t_0, t_1, t_2 is a monotonically increasing sequence of non-negative real numbers which grows without bound in case it is an infinite series.
4. If any action in any class c is enabled in any state s_i of α then within time $s_i.time + t_c$ either some action in c occurs or some state s_j occurs in which every action in c is disabled.

Let α be an execution. Let γ be the subsequence of α consisting of timed external actions, and let t_0 be the start time of α. The *behavior* β corresponding to α is the sequence $\beta = t_0, \gamma$.

1.8 Stabilization: Definition and Properties

Definition based on Execution: Automaton A stabilizes to the executions in set C in time t if

$$\forall \alpha \in A,\ \exists \text{some suffix of } \alpha (\text{which starts within time } t \text{ of} \alpha) \in C$$

Execution stabilization is transitive.
Definition based on External Behavior: Consider two behaviors $\beta = t_0, \gamma$ and $\beta' = t'_0, \gamma'$. We say that β' is a *t-suffix* of behavior β if:

1. $\beta'.start - \beta.start \leq t$
2. γ' is a suffix of γ containing all actions in β that occur at times strictly greater than $\beta'.start$.

Then, an IOA A stabilizes to the behavior sequence set P in time t if for every behavior β of A there is a t-suffix of behavior β that is in P.

1.9 Network Model

It is a standard asynchronous message passing model, with one packet on a link at a time and for every pair of neighbors, one node is apriori declared the "leader" of the pair. The automaton is as shown in Fig 4 of APV paper.

With this background which is common to both papers, we now discuss their respective algorithms.

2 Local Checking and Correction

2.1 Link Subsystem and Local Predicates

A property is said to be local to a subgraph G' for a graph G, if the truth of the property can be ascertained by examining only the components specified by G'. This paper concentrates on *link subsystems* that consist of a pair of neighboring nodes u and v and the channels between them. For any state s, $s|u$ denotes s projected on node N_u and $s|(u,v)$ denotes s projected on link (u,v). For a network automaton N with a link between nodes u and v, if the state is s, then the state of the (u,v) subsystem is the 4-tuple: $(s|u, s|(u,v), s|(v,u), s|v)$.

L is a link predicate set for $N = Net(G,N)$ if for each $(u,v) \in G$ there is some $L_{u,v}$ such that:

1. If $(a,b,c,d) \in L_{u,v}$ then $(d,c,b,a) \in L_{v,u}$.
2. $L = \{L_{u,v}, (u,v) \in G\}$
3. There is at least one state s of N such that s satisfies $L_{u,v} \forall L_{u,v} \in L$ (This is also called Conjunction)

2.2 Local Checkability

A network automaton N is locally checkable for predicate L using link predicate L if:

1. L is a link predicate set for N and L $\in Conj(L)$.
2. Each $L_{u,v} \in L$ is closed.

The second condition is added because in an asynchronous distributed system, it is apparently impossible to check whether an arbitrary local predicate holds all the time.

2.3 Local Correctability

Let us define a local reset function f, with three arguments: a node u, state of node automaton N_u, and neighbor v of node u. Let s be a state of N. It is represented in short by $f(s|u,v)$. If there is a partial ordering $<$ among nodes, then for any state s of N and any edge (u,v) of graph:

- **Correction:** $(f(s|u,v), nil, nil, f(s|v,u)) \in L_{u,v}$
- **Stability:** For any neighbor w of v, if $(s|u, s|(u,v), s|(v,u), s|v) \in L_{u,v}$ and $\{v,w\} < \{u,v\}$ is false then
  $(s|u, s|(u,v), s|(v,u), f(s|v,w)) \in L_{u,v}$.

With such a reset function, a network is locally correctable to L if:

- N is locally checkable for L in L.
- f is a local reset function for N.

Theorem 6.1 Consider any network automaton N that is locally correctable using a link predicate L, local reset function f, and partial order $<$. Then there exists some N^+ that is an unrestricted IOA for constants c and $\tilde{c}$ such that N^+ stabilizes to the behaviors of $N(c)|L$ in time $\tilde{c}.height(<)$.

2.4 Transformation Code

The leader of each (u,v) subsystem will periodically do a *local snapshot* to check if (u,v) subsystem satisfies its predicates; if the leader detects a violation, it tries to make the local predicate true by doing a *local reset* of the (u,v) subsystem. To make the snapshot and reset protocols stabilizing, counter flushing technique is used. (Counter flushing paper has already been covered in Lecture 15 by Prof Patt-Shamir). Please look at Figure 7 in APV paper.

In this technique all request and response packets are numbered. Also, the leader keeps a variable $count_u[v]$ that u uses to number all requests sent to v within a phase. At the end of each phase, the value is incremented. Similarly, the responder v keeps a variable $count_v[u]$ in which v stores the number of the last request it has received from u. Node v weeds out duplicates by only accepting requests with number not equal to $count_u[v]$. Counter flushing guarantees that within 5 phases requests and responses are correctly matched.

The augmented automata has been shown in Figure 8 of APV Paper.

2.5 Intuitive Proof of Local Correction Theorem

2.5.1 Intuition behind Local Snapshot

Please look at Figure 11 of APV paper. To understand why a snapshot works correctly if the response from v is sent following the receipt of the request at u, first let a' and b be the state of nodes u and v respectively just before the response is sent and a,b' be the states just after the response is delivered. Then, it is possible to construct an execution which starts in state (a', nil, nil, b) has an intermediate state (a, nil, nil, b) and has a final state (a, nil, nil, b').

This is because first we apply all changes on node u and change it from a' to a and then apply all changes to v. Thus the state (a, nil, nil, b) recorded by the snapshot is a possible successor of the state of the subsystem when the response is sent. Similarly it is also a possible predecessor of the state when responses are delivered. Since, $L_{u,v}$ is close, we are guaranteed that the detection mechanism will not produce false alarms and will always detect a violation.

2.5.2 Intuition behind Local Resets

Using similar arguments to that of snapshot case, there is an execution sequence possible which means that even though the resets have been applied on u and v when they sent and received response, $L_{u,v}$ still holds at the end of reset phase.

2.5.3 Intuition behind Local Correction Theorem

We can now see why the augmented automaton will ensure that all local predicates hold in time proportional to the height of the partial order. This is because the request-response matching starts from the minimal element in the partial order and once $L_{u,v}$ holds for the subsystem, it remains true. In each 5 phases, new subsystems are added until the local predicate holds in the whole network.

3 Local Checking and Global Reset

In this paper a reset protocol is used to provide, upon request, a consistent signal common to all nodes in the network. The external interface of the protocol at some node u is as shown in Fig. 2 of APVD paper. The external interface between a reset service and its users mimics the link interface with packets replaced by messages (SEND, RECEIVE, FREE to accept another message). The reset service at node, however, offers two extra actions: an input action REQUEST to request a reset and an output action SIGNAL to inform that the reset has been completed at that node.

3.1 Behavior Specifications for Reset Protocols

Response time of the protocol is denoted by R, and *well-formedness* condition is also enforced on the behavior. Formally, a behavior is *well-formed* if between any to SENDs there is a FREE to request a new send. Some more requirements on the reset protocol are:

3.1.1 Timeliness

A behavior is timely if for the whole network the following holds:

1. At any point during the behavior, either $FREE_{u,v}$ occurs within constant time, or else SIGNAL occurs at one of the nodes in O(R) time.
2. Every send is followed by a receive in O(R) time, or else SIGNAL occurs at one of the two corresponding nodes.

3.1.2 Causality

A behavior is causal if the following holds:

1. For every signal event, there is a request event that occurs within the preceding O(R) time.
2. For every request event, there is a signal event that occurs within the following O(R) time.
3. For any signal event, there are signal events that occur at all nodes in the network within the preceding and following O(R) time units.

3.1.3 Mating Relation

Now, we define *signal intervals* at any node u. These are the subsequences of the total behavior demarcated by $SIGNAL_u$ events. If the number of $SIGNAL_u$ events is finite, then u's *final interval* is the infinite interval that begins with the last $SIGNAL_u$ event. Given a behavior, let I_u denote the set of signal intervals at $u \in V$, and let $\hat{I}$ denote the set of all signal intervals. A *mating relation* "$\leftrightarrow$" is a reflexive, symmetric relation over $\hat{I} \times \hat{I}$ that satisfies the following conditions for any node in the link (u, v) in the network:

1. If a receive occurs in I_v and the corresponding send event occurs in I_u, then $I_u \leftrightarrow I_v$.
2. For all values of I_u, there exists at most one I_v, such that $I_u \leftrightarrow I_v$.
3. If $I_u \leftrightarrow I_v$, the sequence of messages v receives from u is a prefix of the messages u sends to v.
4. For all $I_u \leftrightarrow I_v$, I_v is final iff I_u is final. If both are final then all the messages sent have been received.
5. If $I_u \leftrightarrow I_v$ and $I'_u \leftrightarrow I'_v$ then I_v precedes I'_v iff I_u precedes I'_u.

3.1.4 Consistency

A behavior is *weakly consistent* if every RECEIVE action has a corresponding send event, and there exists a mating relation over the set of signal intervals of the behavior. A behavior is *strongly consistent* if the behavior is weakly consistent and in addition the mating relation is transitive.

3.1.5 Internal Reset

A protocol R is a weak(strong) internal reset if any well-formed behavior of R is a suffix of some timely, causal and weakly (resp. strongly) consistent behavior.

The basic protocol of Katz and Perry requires a stable set of paths to a fixed leader and it stabilizes in $O(n^2)$ time while the APV protocol stabilizes in $O(n)$ time.

3.2 Self-stabilization by Local Checking and Global Reset

Theorem 10 Let N be any network automaton that is locally checkable for some predicate Π. Then there exists some constant c, and some uninitialized network automaton N^+ which stabilizes to the behaviors of $N(c)|\Pi$ in $O(R)$ time, where R is the response time of any stabilizing internal reset protocol.

The transformation is same as that described in previous paper, the Counter flushing mechanism.

Analysis The local checking is provenly stabilizing in 5 steps. We assume that we are given a correct internal reset protocol. We need to prove that in $O(R)$ time, the local checking process will stop making reset requests. We know eventually Counter Flushing stabilizes. The mating property can be used to deduce that the corresponding snapshots occur within the same signal interval. Thus each local predicate is satisfied and stabilized eventually and global correct state is achieved.