Delay-Insensitive Stabilization

Anish Arora[1] and Mohamed G. Gouda[2]

[1]Department of Computer and Information Sciences
The Ohio-State University, Columbus, OH 43210

[2]Department of Computer Sciences
The University of Texas at Austin, Austin, TX 78712

Abstract

We consider a class of systems, each of which satisfies the following property of system stabilization: starting from any state, the system will reach a fixed point. We show how to add asynchronous delays to systems in this class, and present several results concerning the effect of adding or removing delays on the property of system stabilization. First, we show that adding or removing delays does change the set of fixed points of a system. Second, we show that removing a delay preserves system stabilization, but adding a delay may not. Third, we identify three types of delays: non-cyclic, short, and long. We show that adding non-cyclic or short delays preserves system stabilization. We also show that under some lax conditions, adding long delays preserves system stabilization.

1 Introduction

A computing system is stabilizing iff starting from an arbitrary, possibly illegitimate state, the system is guaranteed to reach a legitimate state in a finite number of steps. Stabilization is a fundamental property of computing systems. For instance, the reset of a system, the ability of a system to tolerate faults, and the ability of a system to adapt to changes in its environment can all be viewed as special forms of stabilization. See for example [2], [3], [4], and [5].

Unfortunately, stabilization is a "fragile" property. Many seemingly lame transformations of a stabilizing system can disrupt the stabilization of the system [6]. Thus, transformations of stabilizing systems should be allowed only after showing that these transformations do not disrupt system stabilization.

In this paper, we discuss an important class of transformations of stabilizing systems, namely the adding or removing of "asynchronous delays". These transformations are important because any implementation of a system is bound to add such delays to the system or to remove such delays from the system. Although these transformations clearly preserve most properties of the original system, it is not obvious that they do preserve the property of stabilization. The main result of this paper is that removing delays preserves system stabilization but adding delays may not, except in some special cases.

We start our presentation in the next section by identifying a rich class of systems and showing how to add delays to each system in this class.

2 Systems with Delays

A system S consists of some variables x, y, ..., and z, and an equal number of assignment statements of the form

x := F(x, y, ..., z)

y := G(x, y, ..., z)

...

z := H(x, y, ..., z)

where F, G, ..., and H are total functions over the variables in S.

A state of a system S is a mapping that maps each variable in S to a value from the domain of that variable.

A transition of a system S is a triple (p, s, q), where p and q are states of S, s is an assignment statement in S, and executing statement s when S is in state p yields S in state q. For any transition (p, s, q), p is called the tail state of the transition, and q is called the head state of the transition.

A computation of a system S is an infinite sequence of transitions of S such that the following two conditions hold.

i. Order:
In the sequence, the head state of each transition is the same as the tail state of the next transition.

ii. Fairness:
Each assignment statement in S appears infinitely many times in the sequence.

The tail state of the first transition in a computation is called the initial state of the computation. Moreover, if a transition in a computation has a state p (as the tail or head state of that transition), then the computation is said to reach state p.

A state p of a system S is called a fixed point iff each transition (p, s, q) of S is such that p = q.

A system S is stabilizing iff each computation of S reaches a fixed point.

Let x be a variable in a system S. A delay can be added to variable x by modifying system S as follows.

i. Add a new variable dx whose domain of values is the same as that of variable x.

ii. Add an assignment statement of the form dx := x.

iii. Replace each occurrence of x by dx in the assignment statement of every variable, other than x and dx.

In the resulting system, variable x is referred to as a delayed variable, and variable dx is referred to as a delay variable.

In the next section, we discuss an example where delays are added to some stabilizing system in order to facilitate the implementation of this system by a network of communicating processes. In this case, it is important that the resulting system after adding the delays has all the interesting properties of the original system, including the property of system stabilization.

3 Example of a System with Delays

Let S be a system that consists of variables x and y (whose values range over the positive integers), and the two assignment statements

x := **if** x > y **then** x - y **else** x

y := **if** x < y **then** y - x **else** y

It is straightforward to show that each computation of system S reaches a fixed point at which x = y. Therefore, system S is stabilizing.

Delays can be added to the two variables in system S yielding a new system S'. System S' consists of variables x, dx, y, and dy (whose values range over the positive integers), and the four assignment statements

x := **if** x > dy **then** x - dy **else** x

dx := x

y := **if** dx < y **then** y - dx **else** y

dy := y

It is straightforward to show that each computation of system S' reaches a fixed point where x = dx = y = dy. Therefore, system S' is stabilizing.

One advantage of system S' over system S is that system S' can be implemented by a network of two communicating processes. In particular, system S' can be implemented by a network N' of two processes px and py that communicate by exchanging positive integers over two channels cx and cy. Process px sends positive integers to channel cx and receives positive integers from channel cy, whereas process py sends positive integers to channel cy and receives positive integers from cx. Each of the two channels can hold at most one integer at a time. Hence, if a channel holds an integer and a process sends another integer to that channel, then the latter integer replaces the former integer in the channel.

Process px has two local variables vx and d, and two actions. The first action is executed periodically, and its execution causes px to send the current value of variable vx to channel cx. The second action is executed when there is an integer in channel cy, and the execution causes px to receive that integer from cy, store it in variable d, then use the current values of variables vx and d to update the value of vx. Process px is defined as follows.

```
process px
var     vx, d              :     positive integer
begin
        true               -->   send vx to cx
[]      rcv d from cy      -->   vx := if vx > d then vx - d else vx
end
```

Process py is similar to process px and it is defined as follows.

```
process py
var     vy, d           :       positive integer
begin
        true            -->     send vy to cy
[]      rcv d from cx   -->     vy := if vy > d then vy - d else vy
end
```

In order to show that network N implements system S', we need the following definition.

A state (x, dx, y, dy) of system S' is a shadow of a state (vx, cx, vy, cy) of network N iff the following two conditions hold.

i. $x = vx$ and $y = vy$.

ii. If channel cx is empty, then dx = vx, otherwise dx = the integer in channel cx, and
if channel cy is empty, then dy = vy, otherwise dy = the integer in channel cy.

Based on this definition, it is straightforward to show the following relation between the computations of N and those of S'. For each infinite computation of N, there is an infinite computation of S' such that for each reachable state p in the computation of N, there is a corresponding reachable state, denoted C.p, in the computation of S', where the following two conditions hold.

i. State C.p is a shadow of state p.

ii. If a state p occurs before a state q in the computation of N, then state C.p occurs before state C.q in the computation of S'.

From this relation between the computations of N and those of S', and from the fact that each computation of S' has an infinite suffix where x = y at each state of the suffix, we conclude that each computation of network N has an infinite suffix where vx = vy at each state of the suffix. This proves that network N implements system S'.

In this example, we showed that when delays are added to a specific stabilizing system, the resulting system is stabilizing and can be implemented by a network of communicating processes. This example raises the following question: When delays are added to a stabilizing system, is the resulting system guaranteed to be stabilizing? This question is answered negatively in the next section.

4 Stabilization of Systems with Delays

In this section, we discuss two important properties concerning the stabilization of systems with delays. First, we show that adding delays to a system, or removing delays from a system, does not change substantially the set of fixed points of that system. Second, we show that removing delays from a system preserves the property of stabilization but adding delays to a system does not necessarily preserve that property.

Let S be a system and S' be a system that results from adding delays to some variables in system S. A state p of system S and a state p' of system S' are compatible iff the following two conditions hold.

i. For every variable x in system S, the value of x at state p equals the value of of x at state p'.

ii. For every two variables x and dx in system S', the value of x at state p' equals the value of dx at state p'.

Based on this definition of compatible states, we can state the following theorem.

Theorem 1

Let S be a system and S' be a system that results from adding delays to some variables in S. There is a one-to-one correspondence between the fixed points of S and those of S' such that each fixed point of S and the corresponding fixed point of S' are compatible.

[]

According to Theorem 1, systems S and S' have basically the same set of fixed points. However, according to the next theorem, stabilization of system S (to a fixed point) does not guarantee stabilization of S' (to a fixed point).

Theorem 2

Let S be a system and S' be a system the results from adding a delay to some variable in S.

i. If S' is stabilizing, then S is stabilizing.

ii. The converse of i is not necessarily true.

Proof of i

Consider an arbitrary computation C of system S. We need to show that computation C reaches a fixed point of S. This can be accomplished by using computation C to construct a computation C' of system S' such that the following compatibility condition holds. If the reachable states along computation C are p_1, p_2, ..., then computation C' reaches states q_1, q_2, ... such that every two corresponding states p_i and q_i are compatible. Given that S' is stabilizing, computation C' is guaranteed to reach a fixed point of S'. Hence, there is a value k such that each of the states q_k, q_{k+1}, ... is a fixed point of system S'. From the compatibility condition, each of the states p_k, p_{k+1}, ... is a fixed point of system S. Therefore, the arbitrary computation C reaches a fixed point of S.

It remains now to show how to use computation C to construct computation C' such that the above compatibility condition holds. Assume that system S' results from adding a delay dx to variable x in system S. In this case, for each transition (p_i, s_i, p_{i+1}) in computation C add one transition or two consecutive transitions to computation C' according to the following two rules.

i. If s_i is the assignment statement that updates a variable y, other than x, in system S, then add to C' a transition (q_i, s, q_{i+1}), where s is the assignment statement that updates variable y in system S'.

ii. If s_i is the assignment statement that updates variable x in system S, then add to C' two consecutive transitions (q_i, s, q) and (q, s', q_{i+1}), where s and s' are the assignment statements that update variables x and dx, respectively, in system S'.

It is straightforward to show that computation C and the constructed computation C' satisfy the compatibility condition mentioned above.

Proof of ii

Part ii can be proven by exhibiting a stabilizing system S, then showing that if a delay is added to some variable in S, then the resulting system S' is not stabilizing. Consider a system S that consists of variables x and y (whose values are in the domain 0..1), and the two assignment statements

x := y

y := x

At each fixed point of system S, x = y, and system S is stabilizing (to a fixed point).

Now assume that a delay dx is added to variable x in system S. The resulting system S' consists of variables x, dx, and y, and the three assignment statements

x := y

dx := x

y := dx

At each fixed point of system S', x = dx = y, but system S' is not stabilizing (to a fixed point). This is because system S' can cycle indefinitely through the following states, each of which is of the form (x, dx, y), without ever reaching a fixed point:

(0, 0, 1), (1, 0, 1), (1, 0, 0), (1, 1, 0), (0, 1, 0), (0, 1, 1), (0, 0, 1), ...

[]

Clearly, Theorem 2 can be generalized as follows. Let S be a system and S' be a system the results from adding delays to one or more variables in S. If S' is stabilizing, then S is stabilizing, but the converse is not necessarily true.

5 Stabilization of Systems with Non-Cyclic Delays

Theorem 2 in the last section states that in general adding delays to a system can disrupt the stabilization of that system. Thus, it is useful to identify special cases for which adding delays does not disrupt system stabilization. In this section, we introduce the concept of non-cyclic delays and show that adding non-cyclic delays to a system does preserve stabilization of that system.

Let x and y be two variables in a system S. Variable x depends on variable y iff system S has two states p and q that differ only in their values of y and executing the assignment statement of variable x starting at state p and starting at state q yield two different values for variable x.

The dependency graph of a system S is a directed graph whose nodes represent the variables in S and whose directed edges represent the depends on relation on the variables in S. In other words, the set of nodes in the dependency graph of system S is $\{n_x \mid x \text{ is a variable in S}\}$, and the set of directed edges in the dependency graph of S is $\{(n_y, n_x) \mid \text{variable x depends on variable y in S}\}$.

As an example, the dependency graph of system S presented in Section 3 consists of nodes n_x and n_y and four directed edges: a self-loop at node n_x, a self-loop at node n_y, an edge from node n_x to node n_y, and an edge from node n_y to node n_x.

A variable x in a system S is non-cyclic iff the node that corresponds to variable x in the dependency graph G of system S does not occur in any directed cycle in G.

Theorem 3

Let S be a system and S' be a system that results from adding a delay dx to variable x in system S. If S is stabilizing and variable dx is non-cyclic in system S', then S' is stabilizing.

[]

6 Stabilization of Systems with Short and Long Delays

In this section, we introduce the concepts of short and long delays. We show that adding short delays to a system preserves the stabilization of that system. We then identify a rich class of systems, called restricted systems, and show that adding long delays to a restricted system preserves the stabilization of that system (provided that at least one long delay is added to each directed cycle of length two or more in the system).

Let S be a system, and S' be a system that results from adding delays to some variables in S. The variables in system S' can be partitioned into two classes: original variables that correspond to the variables in system S, and delay variables that are added to system S to form system S'.

Let x be a variable in a system S, and let p be a state of S. Variable x is stable at state p iff executing the assignment statement that updates variable x starting at state p keeps state p unchanged. Note that every variable in a system S is stable at each fixed point of system S.

Let S be a system, and S' be a system that results from adding delays to some variables in S. A computation of system S' is short-delayed iff for each transition (p, s, q) in this computation, if s is an assignment statement that updates an original variable in S', then every delay variable in S' is stable at state p. In other words, along any short-delayed computation, the assignment statement of any original variable is executed only when all delay variables are stable.

Let S be a system, and S' be a system that results from adding delays to some variables in S. System S' is stabilizing assuming short delays iff each short-delayed computation of S' reaches a fixed point.

Theorem 4

Let S be a system and S' be a system that results from adding delays to some variables in S. If S is stabilizing, then S' is stabilizing assuming short delays.

[]

A system S is restricted iff every assignment statement x := F in S satisfies the following two conditions.

i. Variable x depends only on itself and one other variable in system S. Thus, function F has at most two arguments x and y, for some variable y in system S.

ii. Function F is idempotent in argument x. Thus,

$$F(x, y) = F(F(x, y), y)$$

A well-delayed version of a system S is a system S' that results from adding delays to some variables in S such that each directed cycle of length two or more in the dependency graph of S' has at least one node that corresponds to a delay variable in S'.

Let S be a system, and S' be a system that results from adding delays to some variables in S. A computation of system S' is long-delayed iff for each transition (p, s, q) in this computation, if s is an assignment statement that updates a delay variable in S', then every original variable in S' is stable at state p. In other words, along any long-delayed computation, the assignment statement of any delay variable is executed only when all original variables are stable.

Let S be a system, and S' be a system that results from adding delays to some variables in S. System S' is stabilizing assuming long delays iff each long-delayed computation of S' reaches a fixed point.

Theorem 5
Let S be a restricted system, and S' be a well-delayed version of system S. If S is stabilizing, then S' is stabilizing assuming long delays.

[]

7 Concluding Remarks

In this paper, we defined how to add asynchronous delays to computing systems and presented several results concerning the effects of adding or removing delays on system stabilization. The presented results are based on the assumption that during any system execution, exactly one assignment statement in the system is executed at a time (interleaving semantics). Nevertheless, similar results can be obtained based on the assumption that during any system execution, any subset of the assignment

statements in the system are executed at a time (powerset semantics in [1] and [7]).

References

[1] A. Arora, P attie, M. Evangelist, and M. Gouda, "Convergence of Iteration Systems", Distributed Computing, Volume 7, pp. 43 - 53, 1993.

[2] S. Dolev and T. Herman, "Superstabilizing Protocols for Dynamic Distributed Systems", Proceedings of the Second Workshop on the Self-Stabilizing Systems, UNLV, pp. 3.1 - 3.15, 1995. Also, Short Abstract in Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing, p. 255, 1995.

[3] S. Dolev, A. Israeli, and S. Moran, "Self-Stabilization of Dynamic Systems Assuming only Read Write Atomicity", Distributed Computing, Volume 7, pp. 3 - 16, 1993.

[4] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju, "Fault-Containing Self-Stabilizing Algorithms", Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, p. 45 - 54, 1996.

[5] M. G. Gouda, "The Triumph and Tribulation of System Stabilization", Invited Paper in the 9th International Workshop on Distributed Algorithms, 1995. Also, appeared in Lecture Notes in Computer Science, Volume 972, pp. 1 - 18, 1995.

[6] M. G. Gouda, R. R. Howell, L. E. Rosier, "The Instability of Self-Stabilization", Acta Informatica, Vol. 27, pp. 697 - 724, 1990.

[7] F. Robert, "Discrete Iterations - a Metric Study", Springer-Verlag, Berlin, 1986.