Brief Announcement: The Asynchronous Bounded-Cycle Model

Peter Robinson and Ulrich Schmid

Technische Universität Wien, Embedded Computing Systems Group (E182/2), Treitlstrasse 1-3, A-1040 Vienna (Austria)
{robinson,s}@ecs.tuwien.ac.at

Introduction

Recent research[1] on fault-tolerant distributed clock generation in VLSI circuits [2] revealed that the Θ-Model introduced in [4, 7] is well-suited for this promising application domain of distributed algorithms. The Θ-Model bounds the ratio between the maximal and minimal end-to-end delay of messages simultaneously in transit between correct processes. Its suitability for VLSI applications primarily results from the fact that placement & routing of the functional units in a VLSI chip offers some control about Θ, whereas technology scaling, which severely affects the end-to-end delays, leaves their ratio Θ essentially unaffected [8].

In this paper, we introduce a novel *Asynchronous Bounded-Cycle* (ABC) model, which relaxes the Θ-Model even further: It (1) avoids any reference to end-to-end delays and hence applies to purely asynchronous executions, (2) allows individual messages to have arbitrary end-to-end delays, and (3) does not involve global synchrony conditions but acts on some causally related messages only. Essentially, there is only one scenario that is not admissible in the ABC model: A chain C_1 of k_1 consecutive messages, starting at process p and ending at q, that properly "spans" (see Fig. 3) another causal chain C_2 from p to q involving $k_2 \geqslant k_1\Xi$ messages, for some model parameter $\Xi > 1$. In practice, this implies that the *cumulative* message delays along C_2 must not become so small that C_1 could span $k_1\Xi$ or more messages in C_2. Algorithms can exploit the fact that this property allows to "time out" certain message chains, and hence failure detection.

Contribution

We consider a system of n distributed processes, connected by a (not necessarily fully-connected) point-to-point network with finite but unbounded message delays. We neither assume FIFO communication channels nor an authentication service, but we do assume that processes know the sender of a received message. Every process executes an instance of a distributed algorithm and is modeled as a state machine. Its local execution consists of a sequence of atomic zero-time computing steps, each involving the reception of exactly one message, a state transition, and the sending of zero or more messages to a subset of the processes in the system. Among the n processes in the system, at most f may be Byzantine faulty.

The ABC model just puts one additional constraint on admissible executions, i.e., executions where every correct process makes infinitely many computing steps and every message sent by a correct process is received by every recipient within finite time. It is solely based on the space-time diagram [3], which captures the causal flow of information.

Definition 1 (Effective execution graph). *The* effective execution graph G *is the digraph corresponding to the space-time diagram of an admissible execution α, with nodes $V(G) = \Phi$ corresponding to the receive events in α, and edges reflecting the happens-before relation [3]: (ϕ_i, ϕ_j) is in the edge relation $\rightarrow : \Phi \times \Phi$ iff one of the following two conditions holds:*

1. *The receive event ϕ_i triggers a computing step in which a message m is sent from a correct process p to some (possibly faulty) process q; event ϕ_j is the receive event of m at q. We call the edge $\phi_i \rightarrow \phi_j$* non-local edge *or simply* message *in G.*
2. *The events ϕ_i and ϕ_j both take place at the same processor p and there exists no event ϕ_k in α occurring at p such that $i < k < j$. The edge $\phi_i \rightarrow \phi_j$ is said to be a* local edge.

[1] Supported by the Austrian FWF projects *Theta* (P17757) and PSRTS (P20529), and the Austrian bm:vit FIT-IT project *DARTS* (809456-SCK/SAI), see *http://ti.tuwien.ac.at/ecs/research/projects/darts/* for details.

A *causal chain* $\phi_1 \to \cdots \to \phi_l$ is a directed path in the execution graph. Note that a causal chain may contain local edges. The *length of a causal chain* D is the number of non-local edges (i.e., messages) in D, denoted by $|D|$. A *cycle* Z in G is a subgraph of G that corresponds to a cycle in the undirected shadow graph of G. Since messages cannot be sent backwards in time, every cycle can be decomposed into at least 2 causal chains having opposite directions.

Definition 2 (Relevant cycles). *Let Z be a cycle in the effective execution graph, and partition the edges of Z into the* backward *edges $\hat{Z}^-$ and the* forward *edges $\hat{Z}^+$ as follows: Identically directed edges are in the same class, and $|Z^+| \leqslant |Z^-|$, where $Z^- \subseteq \hat{Z}^-$ and $Z^+ \subseteq \hat{Z}^+$ denote the sets of non-local edges (i.e., messages) having the same direction. The* orientation *of the cycle Z is the direction of the forward edges Z^+, and Z is said to be* relevant *if it does not contain any forward* local *edges, i.e., if $\hat{Z}^+ = Z^+$.*

Definition 3 (ABC synchrony condition). *Let $\Xi > 1$ be a given rational number, and let α be an admissible execution in the ABC model. Then, for every relevant cycle Z in the effective execution graph G_α, it holds that*

$$\frac{|Z^-|}{|Z^+|} < \Xi. \tag{1}$$

Figure 3 shows an example of a relevant cycle. Note carefully that, relative to the purely asynchronous model, there is no other constraint in the ABC model. Nevertheless, the ABC synchrony condition is sufficient for clock synchronization and simulating lock-step rounds by means of the following algorithms:

Algorithm 1: Byzantine Clock Synchronization

```
1: VAR k: integer ← 0;
2: send (tick 0) to all [once];

   /* catch-up rule */
3: if received (tick l) from f + 1 distinct processes
   and l > k then
4:     send (tick k + 1),...,(tick l) to all [once];
5:     k ← l;

   /* advance rule */
6: if received (tick k) from n − f distinct processes then
7:     send (tick k + 1) to all [once];
8:     k ← k + 1;
```

Algorithm 2: Lock-Step Round Simulation

```
1: VAR r: integer ← 0;
2: call start(0);

3: Whenever k is updated do
4: if k/(2Ξ) = r + 1 then
5:     r ← r + 1
6:     call start(r)

7: procedure start(r:integer)
8:     if r > 0 then
9:         read round r − 1 messages
10:        execute round r computation
11:    send round r messages
```

In the clock synchronization Algorithm 1, every process p maintains a local variable k that constitutes p's local clock:[2] Every process initially sets $k \leftarrow 0$ and broadcasts the message (*tick* 0). If a correct process p receives $f+1$ (*tick* ℓ) messages (catch-up rule, line 3), it can be sure that at least one of them was sent by a correct process that has already reached clock value l and therefore can safely catch-up to l and broadcast (*tick* $k+1$),...,(*tick* l). If some process p receives $n - f \geqslant 2f + 1$ (*tick* k) messages (advance rule, line 6) and thus advances its clock to $k+1$, it follows that at least $f+1$ of those messages will also be received by every other correct process, which then executes line 3. Hence, all correct processes will eventually receive $n - f$ (*tick* k) messages and advance their clock to $k+1$.

Theorem 1 shows that the clocks of correct processes remain synchronized. The main idea of the proof is that the distance between clock values corresponds to the number of backward messages in a relevant cycle, and is therefore bounded by (1), cp. Fig. 4. Note that the ABC model—being entirely time-free—actually guarantees a synchrony property that can be stated in terms of consistent cuts, which has been translated to real-time cuts according to [5] to derive Theorem 1.

[2] For simplicity, the pseudo-code of our algorithm assumes that a process sends messages to all processes including itself. In reality, however, these self-receptions are just part of the local computation.

Theorem 1 (Precision on Clocks). *Let $C_p(t)$ denote the clock value of process p at time t. For any time t of an admissible execution of Algorithm 1 in a system with $n \geqslant 3f+1$ processes, we have $|C_p(t) - C_q(t)| \leqslant 2\Xi$ for all correct processes p, q.*

To guarantee lock-step rounds (Theorem 2), we use the same simulation as in [8], where clocks are phase counters and rounds consist of 2Ξ phases. Algorithm 2 shows the code that must be merged with Algorithm 1; all round r messages are piggybacked on (*tick* k) messages every 2Ξ phases, namely, when $k/(2\Xi) = r + 1$. The round r computing step is encapsulated in the function start(r) in line 7.

Theorem 2 (Lock-Step Rounds). *In a system with $n \geqslant 3f + 1$ processes, Algorithm 2 merged with Algorithm 1 correctly simulates lock-step rounds in the ABC model.*

Finally, we proved that all algorithms designed and proved correct for the Θ-Model also work correctly in the ABC model, despite the fact that (most) ABC executions are not admissible in the Θ-Model. The proof is based on a novel technique for assigning message delays to asynchronous executions, which involves a non-standard cycle-space of a graph and an algebraic treatment of a system of linear inequalities using a variant of Farkas' lemma.

Theorem 3 (Model Indistinguishability). *If an algorithm satisfies a property P in the ABC model, it also satisfies P in the Θ-model for $\Theta < \Xi$. Conversely, all timing-independent properties satisfied by an algorithm in the Θ-model also hold in the ABC model for $\Xi < \Theta$.*

The ABC model, along with the algorithms designed for the Θ-Model, is hence excellently suited for being applied in the VLSI context, as well as in other applications where end-to-end delays can (jointly) vary and even grow continuously with time. Note that Theorem 3 also implies that most of the findings of the very detailed relation of the Θ-Model to other partially synchronous models (in particular, DLS [1]) provided in [8] also apply a fortiori to the ABC model. A full version of our paper is available as [6].

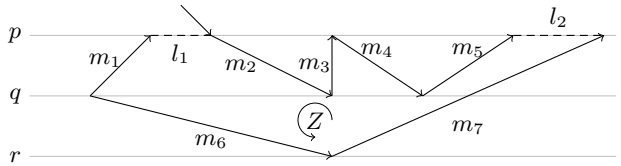

Fig. 3: A relevant cycle Z where a causal chain $C_2 = m_1 l_1 m_2 \ldots m_5 l_2$ is properly spanned by the "slow" chain $C_1 = m_6 m_7$. Message m_3 has zero delay.

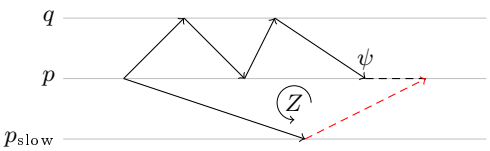

Fig. 4: Let $\Xi = 2$. If the reply message arrived from p_{slow} at p after event ψ, there would be a relevant cycle Z where $\frac{|Z^-|}{|Z^+|} = \frac{4}{2} \not< \Xi$, a contradiction.

References

1. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer, *Consensus in the presence of partial synchrony*, Journal of the ACM **35** (1988), no. 2, 288–323.
2. Matthias Fuegger, Ulrich Schmid, Gottfried Fuchs, and Gerald Kempf, *Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip*, Proceedings of the Sixth European Dependable Computing Conference (EDCC-6), IEEE Computer Society Press, October 2006, pp. 87–96.
3. Leslie Lamport, *Time, clocks, and the ordering of events in a distributed system*, Commun. ACM **21** (1978), no. 7, 558–565.
4. Gérard Le Lann and Ulrich Schmid, *How to implement a timer-free perfect failure detector in partially synchronous systems*, Tech. Report 183/1-127, Department of Automation, Technische Universität Wien, January 2003.
5. Friedemann Mattern, *On the relativistic structure of logical time in distributed systems*, 1992.
6. Peter Robinson and Ulrich Schmid, *The asynchronous bounded-cycle model*, Research Report 24/2008, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2008.
7. Josef Widder, Gérard Le Lann, and Ulrich Schmid, *Failure detection with booting in partially synchronous systems*, Proceedings of the 5th European Dependable Computing Conference (EDCC-5) (Budapest, Hungary), LNCS, vol. 3463, Springer Verlag, April 2005, pp. 20–37.
8. Josef Widder and Ulrich Schmid, *Achieving synchrony without clocks*, Research Report 49/2005, Technische Universität Wien, Institut für Technische Informatik, 2005, (submitted).