BG Distributed Simulation Algorithm

Matthieu Roy

▶ **To cite this version:**

Matthieu Roy. BG Distributed Simulation Algorithm. Ming-Yang Kao. Encyclopedia of Algorithms, Springer New York, pp.199-203, 2016, 978-1-4939-2863-7. 10.1007/978-1-4939-2864-4_611 . hal-02087581

HAL Id: hal-02087581
https://hal.science/hal-02087581

Submitted on 2 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Title:	BG simulation algorithm
Name:	Matthieu Roy
Affil./Addr.	LAAS; CNRS, Univ. Toulouse, France
Keywords:	Read/Write Shared Memory; Reduction; Computability; Distributed Tasks
SumOriWork:	1993; Borowsky, Gafni 2001; Borowsky, Gafni, Lynch, Rajsbaum

BG simulation algorithm

Matthieu Roy
LAAS; CNRS, Univ. Toulouse, France

Years aud Authors of Summarized Original Work

1993; Borowsky, Gafni
2001; Borowsky, Gafni, Lynch, Rajsbaum

Keywords

Read/Write Shared Memory; Reduction; Computability; Distributed Tasks

Problem Definition

How to effectively translate an algorithm from a distributed system model to another one?

Distributed systems come in diverse settings, that are modeled by different assumptions 1) on the way processes communicate, e.g., using shared memory or messages, 2) on the fault model, 3) on synchrony assumptions, etc. Each of these parameters has a dramatic impact on the computing power of the model, and in practice, an algorithm or an impossibility result is usually tailored to a particular model and cannot be directly reused in another model.

This wide variety of models has given rise to many different impossibility theorems and numerous algorithms for many of the possible combinations of parameters that characterize them. Thus, a crucial question is the following: are there bridges between some models, i.e., is it possible to transfer an impossibility result or an algorithm from one model to another?

The Borowsky-Gafni simulation algorithm, or BG simulation, is one of the first step towards direct translations of algorithms or impossibility results from one model to another. The BG simulation considers distributed systems made of asynchronous processes that communicate using a shared memory array. In a nutshell, this simulation allows a set of $t+1$ asynchronous sequential processes, where up to t of them can stop during their execution, to simulate any set of $n \geq t+1$ processes executing an algorithm that is designed to tolerate to up to t fail-stop failures.

The BG simulation has been used to prove solvability and unsolvability results for crash-prone asynchronous shared memory systems, paving the way for a more

generic formal theory of reduction between problems in different models of distributed computing.

The BG simulation algorithm is named after his authors, Elizabeth Borowsky and Eli Gafni, that introduced it as a side tool [3] in order to generalize the impossibility result of solving a weakened version of consensus, namely k-set agreement [6]. It has been later on formalized and proven correct [4, 17] using the I/O automata formalism [18].

System model

Processes. The simulation considers a system made of up to n asynchronous sequential processes that execute a distributed algorithm to solve a given colorless decision task, as defined below.

Failure model. Processes may fail by stopping (crash failure). The simulation assumes that up to t processes can stop during the execution; $t < n$ is known before the execution, but the identity of processes that may crash is unknown to the simulation. This model of computation is referred to as the t-resilient model. A corner case of this model is the wait free model where $t + 1$ processes execute concurrently and at most t of them may crash.

Communication. Processes communicate and coordinate using a reliable shared memory composed of n multiple-reader single-writer registers. Each process has the exclusive write-access to one of these n registers, and processes can read all entries by invoking a *snapshot* operation, with the semantics that write and snapshot operations appear as if they are executed atomically. While using the snapshot abstraction eases the presentation of the algorithm, it has no impact on the power of the underlying computing model, since the snapshot/write model can be implemented wait-free using read/write registers [1].

Tasks A colorless task is a distributed coordination problem in which every process p_i starts with a value, communicates with other processes, and has to decide eventually on a output value. Colorless tasks, or convergence tasks [12], are a restricted version of tasks in which a deciding process may adopt the decision value of any process, i.e. two participating processes may decide the same value. For more formal definitions of tasks using tools from algebraic topology, the reader should refer to [11]

Simulation The simulation proceeds by executing concurrently, using $t+1$ simulators $s_1, \ldots, s_{t+1}$, the code of $n > t$ processes that collaboratively solve a distributed colorless task. Hence, each simulator s_i is given the code of all simulated processes and handles the execution of n threads.

Key Results

Simulation of memory Each one of the $t + 1$ simulators s_i executes the sequential code of the n simulated processes p_j in parallel. By assumption, every simulated code is a sequence of instructions that are either (1) local processing, (2) a write operation into memory or (3) a snapshot of the shared memory.

Every simulator s_i maintains its local view of the simulated memory for all simulated threads. These local views are synchronized between simulators by writing and reading (using snapshots) in a shared memory matrix array MEM that has one column per simulated thread and one row per *snapshot* instance.

To ensure global consistency between simulators that simulate concurrently all threads, operations on the memory must be coordinated between different simulators.

This is achieved by ensuring that, for a given simulated thread, the sequence of snapshots of the memory as computed by all simulators is identical. As consensus cannot be implemented wait-free, the simulation coordinates snapshots using of a weaker form of agreement, the *safe-agreement*.

The safe-agreement object. Safe-agreement is the most important building block of the simulation. First introduced as the *non-blocking-busy-wait* agreement protocol [3], it has been further refined as safe-agreement, with several blocking or non-blocking/wait-free implementations [2, 14, 11].

This weak form of agreement provides two methods to processes: *propose*(v) and *resolve*(). A participating process that proposes a value v first calls *propose*(v) once, and is then allowed to make calls to *resolve*(), that may return $\perp$ if safe-agreement is not resolved yet, or a value. In this later case, safe-agreement is said to be resolved and the value returned is the decided value by the process. Formally, safe-agreement is defined by three properties:

Termination: If no process crashes during the execution of *propose*(), then all processes decide, i.e. eventually all calls to *resolve*() return a non-$\perp$ value,
Validity: All processes that decide must decide a proposed value,
Agreement: All processes that decide must decide the same value.

The specification is almost identical to the one of consensus, apart from the weakened termination property. Safe-agreement is wait-free solvable, and thus solvable in t-resilient systems.

The crucial point of the BG simulation lies in the termination property of safe-agreement: if a safe-agreement protocol cannot be resolved, i.e. if no process decides, then at least one process crashed during the call to *propose*(). Thus, a given safe-agreement instance can "capture" a calling process that crashed during the *propose* invocation.

Overview of the simulation The current state of the simulation and its history is thus represented by two twin data structures: 1) the shared memory matrix *MEM* contains the consecutive memory status of all simulated threads, as seen by simulators, and 2) a matrix of safe agreement objects SafeAgreement$[0..][1..n]$ with n columns, each column representing the execution advancement of one of the simulated processes, as shown in Figure 1.

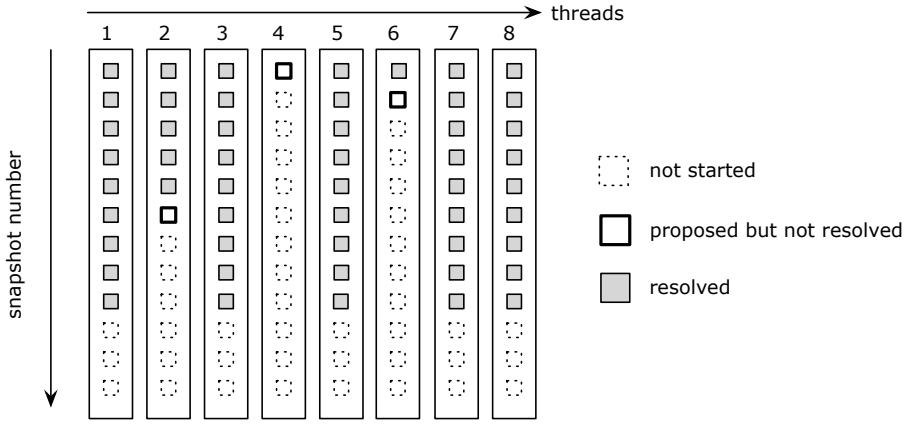

Fig. 1. Conceptual view of advancement for snapshots of all simulated process with $n = 8$ and $t = 3$

In this view, the entry at row ℓ and column i corresponds to the state of the ℓth snapshot for simulated process p_j. Hence, the "program counter" of a simulated thread p_i is the greatest row of column i that is either unresolved or resolved. In this example,

simulations of threads p_2, p_4 and p_6 are stopped with unresolved safe-agreement, that are due to (at least) one simulator stuck in the associated *propose*() methods. Program counters of all other threads is 9.

Each simulator s_i is given the code of the n threads it has to simulate, as well as an input value of one of the threads. Conceptually, the algorithm run by simulator s_i is as follows:

Algorithm 1 BG-simulation: code for a simulator s_j starting with input v

```
 1: procedure BG-SIMULATION(v)
 2:     ∀i = 1..n, SafeAgreement[0][i].propose(v)                ▷ Initialization
 3:     loop
 4:         for i ← 1, n do                                      ▷ Simulate threads in round-robin
 5:             ℓ ← current program counter of p_i
 6:             snap ← SafeAgreement[ℓ][i].resolve()
 7:             if snap ≠ ⊥ then                                 ▷ safe agreement is resolved
 8:                 perform local computation using snap, write operations in local memory
 9:                 execute write on behalf of p_i in MEM[ℓ][i]
10:                 if thread p_i is terminated then
11:                     return value and stop its simulation
12:                 else if at least (n − t) threads have program counter ≥ ℓ then
13:                     snap ← snapshot(MEM[ℓ])
14:                     SafeAgreement[ℓ + 1][i].propose(snap)
15:                 end if
16:             end if
17:         end for
18:     end loop
19: end procedure
```

In the simulation, each *snapshot* invocation is mediated through a SafeAgreement object, lines 6 and 14. The only reason that could block the simulation of a given thread p_i is when the call to *resolve*, line 6, always returns $\perp$. By definition of the safe-agreement object, this situation can happen only when a simulator crashed during the call to *propose*() on the same safe-agreement instance: *the crash of a simulator can block the simulation of at most one simulated thread.*

Applications

The BG-simulation algorithm has been primarily used to reduce t-resilient solvability to wait-free solvability for colorless tasks, that is tasks that are agnostic on process identities. The initial application has been made to the k-set agreement problem, in which all processes have to agree on a final set of values of size at most k. If k-set agreement was solvable in a k-resilient system of $n > k + 1$ processes, then the BG-simulation of this algorithm with $k+1$ simulators would produce a wait-free solution to k-set agreement. Since k-set agreement is not wait-free solvable for $k+1$ processes [19, 13], it follows a contradiction.

The BG-simulation presented here only applies to *colorless tasks*. Gafni [8] extended further to more general classes of tasks, and provided the general characterization of t-resilient solvable tasks, similarly to the Herlihy-Shavit conditions for wait-free computability [13]. This extension has been also studied in [14, 16].

In order to study the relationship between wait-freedom and t-resilience, [5] uses objects of type $\mathcal{S}$ in addition to read-write registers, and shows that for any $t < k$, t-resilient k-process consensus can be implemented with objects of type $\mathcal{S}$ and registers if

and only if wait-free $(t+1)$-process consensus can be implemented with objects of type $\mathcal{S}$ and registers. [15] considers models equipped with registers and consensus objects, and extends the results provided by BG simulation, showing equivalences between models based on the ratio between the maximum number of failures and the consensus number of consensus objects.

Chaudhuri and Reiners [7] use BG simulation to provide a characterization of the Set Consensus Partial Order, a refinement of Herlihy's consensus-based wait-free hierarchy [10]; a formal definition of *set consensus number* and a study of associated respective computing power has been later provided in [9].

Recommended Reading

1. Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, September 1993.
2. Hagit Attiya. Adapting to point contention with long-lived safe agreement. In *Proceedings of the 13th International Conference on Structural Information and Communication Complexity*, SIROCCO'06, pages 10–23, Berlin, Heidelberg, 2006. Springer-Verlag.
3. Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 91–100, New York, NY, USA, 1993. ACM.
4. Elizabeth Borowsky, Eli Gafni, Nancy Lynch, and Sergio Rajsbaum. The BG distributed simulation algorithm. *Distributed Computing*, 14(3):127–146, 2001.
5. Tushar Chandra, Vassos Hadzilacos, Prasad Jayanti, and Sam Toueg. Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies (extended abstract). In *PODC '94: Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*, pages 334–343, New York, NY, USA, 1994. ACM.
6. Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, July 1993.
7. Soma Chaudhuri and Paul Reiners. Understanding the Set Consensus Partial Order Using the Borowsky-Gafni Simulation (Extended Abstract). In *Proceedings of the 10th International Workshop on Distributed Algorithms*, pages 362–379, London, UK, 1996. Springer-Verlag.
8. Eli Gafni. The extended BG-simulation and the characterization of t-resiliency. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 85–92, New York, NY, USA, 2009. ACM.
9. Eli Gafni and Petr Kuznetsov. On set consensus numbers. *Distributed Computing*, 23(3-4):149–163, 2011.
10. Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, January 1991.
11. Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013.
12. Maurice Herlihy and Sergio Rajsbaum. The decidability of distributed decision tasks (extended abstract). In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 589–598, New York, NY, USA, 1997. ACM.
13. Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.
14. Damien Imbs and Michel Raynal. Visiting gafni's reduction land: From the bg simulation to the extended bg simulation. In *SSS*, pages 369–383, 2009.
15. Damien Imbs and Michel Raynal. The multiplicative power of consensus numbers. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, PODC '10, pages 26–35, New York, NY, USA, 2010. ACM.
16. Petr Kuznetsov. Universal model simulation: Bg and extended bg as examples. In *SSS*, pages 17–31, 2013.
17. Nancy Lynch and Sergio Rajsbaum. On the Borowsky-Gafni Simulation Algorithm. In *Proceedings of the of the Fourth Israel Symposium on Theory of Computing and Systems*, ISTCS '96, pages 4–15. IEEE Computer Society, June 1996.
18. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 1996.

19. Michael Saks and Fotios Zaharoglou. Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.