Game Theory and Distributed Algorithms

Mario R. F. Benevides
Carla A. D. M. Delgado

COPPE-Sistemas – Universidade Federal do Rio de Janeiro (UFRJ)
Caixa Postal 68.511 – 21.945-970 – Rio de Janeiro – RJ – Brazil
{mario, delgado}@cos.ufrj.br

Abstract

In this work, we model the consensus problem as an extensive game and prove correctness and termination of the algorithm, relating these properties with Nash equilibrium in the game. Our aim is to use concepts from Game Theory to model distributed algorithms and establish desirable properties.

1 Introduction

A distributed algorithm can be seem as a set of sequential algorithms which run concurrent and independent, each one with his own information. In order to perform a global task, each algorithm has to execute some actions and exchange information by message passing through communication channels.

There exists a large number of applications of distributed algorithms. For instance, services on the internet, protocols for data communication, airline reservation systems, and real-time process control. In all these applications, it is critical to ensure that the algorithms run correctly.

An important class of distributed algorithms is the class of algorithms for consensus. It consists of a group of processes which, by means of information exchange, have to agree on the value of some variable. The communication channels are not reliable but can loose a maximum of M messages.

In this work, we model the consensus problem as an extensive game and prove correctness and termination of the algorithm, relating these properties with Nash equilibrium in the game. Our aim is to use concepts from Game Theory to model distributed algorithms and establish desirable properties.

2 Distributed Systems

This section introduces all the concepts needed about Asynchronous Distributed Systems (ADS). A Distributed System is a system composed of a set of processes that do not share any memory and can communicate only by sending and receiving messages along a

network previously defined. For an asynchronous distributed system, there is no global clock and the delivery time of messages is finite but unbounded. The description of such kind of distributed systems is based on the behaviors of the individual components (also called nodes, agents or processes) in the system. We will now introduce an event-based formalism to describe the distributed computations that take place in ADS, according to the model described in [LL78].

Definition 2.1: Protocol
Is a distributed algorithm, possibly not deterministic, that specifies the actions each agent takes in response to receiving a message.

Definition 2.2: Event
Is a tuple $\xi=<n_i, t, \varphi, \sigma, \sigma', \Phi>$ where:
n_i is the node where the event occurs;
t is the time that occurred the event, given by n_i's local clock.

- φ is the message, if any, received in n_i that triggered the event;
- σ is local state before the event occurrence;
- σ' is local state after the event occurrence;
- Φ is the set of messages, if any, sent by n_i in response to the event occurrence;

Events are the basic unity of time in the asynchronous model. A "send message" event by a node implies a "receive message" event at the target node.

Definition 2.3: Distributed Computation or Execution
Is the execution of a distributed algorithm, the parallel execution of all processes involved in the protocol. A distributed computation can be thought as a set of events Ξ. Σ_i is the state sequence that a process n_i goes through as Ξ evolves. The first member of Σ_i is the initial state, and the last, the final state. Each state change is a consequence of an event (of Ξ) occurrence to process n_i.

Definition 2.4: Temporal relation π
Consider two events v_1 and v_2, $v_1 \pi v_2$ if and only if:

- Both v1 and v2 occurs at the same node, respectively at instants t1 and t2 so that $t1 < t2$. No event v' occurs at the same node at an instant t so that $t1 < t < t2$.
- The events v1 and v2 occurs respectively in processes ni, nj such that a message φ is sent from ni to nj in v1 and received by nj in v2.

The meaning of π is "v_1 happened immediately before v_2", and only makes sense if v_1 and v_2 are events on the same execution.

Definition 2.5: Temporal relation π+:
π^+ is the transitive and irreflexive closure of π, π^+ establishes a partial ordering over the set of events Ξ. Two events v_1 and v_2 which are not ordered by π^+ ($(v_1, v_2) \in \Xi \times \Xi - \pi^+$ and $(v_2, v_1) \in \Xi \times \Xi - \pi^+$) are called concurrents.

Definition 2.6: Future and Past

We can use the relation π^+ to define the future and past of an event ξ in relation to an execution Ξ: Past(ξ) =$\{ \xi' \in \Xi \mid \xi' \pi^+ \xi \}$ and Future(ξ) =$\{ \xi' \in \Xi \mid \xi \pi^+ \xi' \}$

Definition 2.7: System State
Is a collection of local states, one for each node, and an "edge state" for each communicate channel between each node.

Definition 2.8: Consistent Global State or Global State
The global state must reflect the local state of each agent, plus the set of messages in transit. It is said consistent if there are no messages from future to past. Bellow, two equivalent definitions of global estate are presented:

Definition 2.8.1: Definition of global state using total orders on the set of events
In order to define a consistent global state, it's necessary to establish a total order < over Ξ consistent with π^+. Pairs of consecutive events $(\xi_1, \xi_2) \in <$ if for all event $\xi \neq \xi_1, \xi_2$ either $\xi < \xi_1$ or $\xi_2 < \xi$. There is a system state associated to each pair (ξ_1, ξ_2) of consecutive events in < denoted by system_state(ξ_1, ξ_2) with the following properties:

- for a node ni, σi is the resulting state from the occurrence of the most recent event in ni, for example ξ, which is not the case that ($\xi 1 > \xi$).
- for each channel (ni, nj) Φij is the set of messages sent in connection with an event ξ such that it is not the case that ($\xi 1 > \xi$) and received in connection with an event ξ' such that it is not the case that ($\xi' > \xi 2$).

A system state Ψ is global if and only if either all nodes are in their initial states and all channels are empty, or all nodes are in their final states and all channels are empty, or there is a total order < consistent with π^+ for which there is a pair (ξ_1, ξ_2) of events such that Ψ = system_state(ξ_1, ξ_2).

Definition 2.8.2: Definition of global state using partitions on the set of events
A system state Ψ is global if and only if it is represented by a partition (Ξ_1, Ξ_2) of Ξ such that: Past(ξ) $\subseteq \Xi_1$ every time that $\xi \in \Xi_1$ or Future(ξ) $\subseteq \Xi_2$ every time that $\xi \in \Xi_2$.. The partitions that follow this restriction are called consistent cuts. This way, Ψ = system_state(Ξ_1, Ξ_2). The system state represented by the partition (Ξ_1, Ξ_2) when it is a consistent cut is:

- For each ni, its local state is the resulting state from the execution of the most recent event from $\Xi 1$ occurring in ni;
- For each channel (ni, nj), a set of messages sent by ni in events of $\Xi 1$ and received by nj in events of $\Xi 2$.

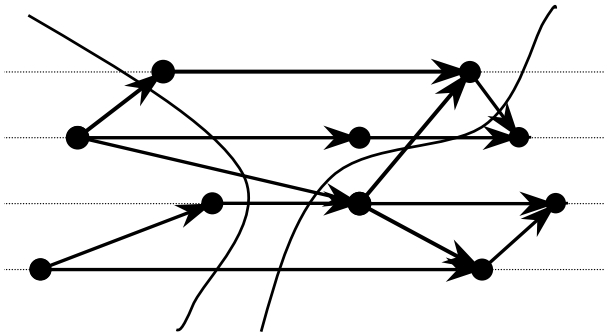


Figure 1: A precedence graph is presented with two partitions. The first corresponds to a consistent cut, whereas the second does not, since there is an edge from future to past.

Definition 2.9: Future and Past of a global state

We can now define future and past of a global state Ψ as: $Past(\Psi) = \cup_{\xi \in \Xi 1} [\{\xi\} \cup Past(\xi)]$ and $Future(\Psi) = \cup_{\xi \in \Xi 2} [\{\xi\} \cup Future(\xi)]$, where $\Psi = system\_state(\Xi_1, \Xi_2)$.

We say that a global estate Ψ_1 comes before another global state Ψ_2 in a computation Ξ if and only if: $Past(\Psi_1) \subset Past(\Psi_2)$ or $Future(\Psi_2) \subset Future(\Psi_1)$.

Definition 2.10: Past view

Given a distributed computation Ξ, the past view of a process n_j considering the global state s given by the partition (Ξ_1, Ξ_2) is the set if events that occurs to process n_j in the past of ξ_j, where ξ_j is the event associated with n_j's local state, that means, ξ_j is the most recent event of Ξ_1 occurring to n_j at the global state s: $Vp(n_j, s) = \{\xi_j' \in \Xi_1 \mid \xi_j' \, \pi^+ \, \xi_j\}$

Definition 2.11: Future view

Given a distributed computation Ξ, the future view of a node n_j considering the global state s given by the partition (Ξ_1, Ξ_2) is the set if events that occurs to agent n_j in the future of ξ_j, where ξ_j is the event associated with n_j's local state, that means, ξ_j is the most recent event of Ξ_1 occurring to n_j at the global state s: $Vf(n_j, s) = \{\xi_j' \in \Xi_2 \mid \xi_j \, \pi^+ \, \xi_j'\}$

3 Extensive Games

This section is based on [OR94], it contains all the concepts need about extensive games.

An extensive game is a game model where different stages of a game can be represented as intermediate game positions, nodes, in a game tree.

A *history* is sequence (finite or infinite) of actions h={a1, a2, …}, and h|k={a1, a2,…,ak} denote the initial sequence of length k of h.

An *extensive game of perfect information* G is defined as

$G=(N,H,P, (\succcurlyeq_i))$, for each $i \in N$. Where

N is the set of players.

H is the set of histories of the game, and must satisfies the following conditions:

1. the empty sequence () $\in$ H.
2. H is closed under initial subsequence.
3. if all finite initial subsequences of an infinite sequence h are in H, the so is h.

P is the player function which assigns to every nonterminal history the player whose turn it is to move.

Finally, ($\succcurlyeq_i$) is a preference relation, one for each player i $\in$ N.

An extensive game is *finite* if H is finite.

Let k be a history of length k, we denote by (h,a) the sequence of length k+1, and A(h)={x $\in$ A | (h,x) $\in$ H}, where A is the set of actions.

A *strategy* for player i is a function which assigns an action in A(h) to each nonterminal history h for which P(h)=i. Given a strategy profile s, we define the outcome O(s) of s to be the history which results when the players use their respective strategies.

Given an extensive game G=(N,H,P, ($\succcurlyeq_i$)), a strategy profile s is a *Nash equilibrium* iff $\forall$i $\in$ N $\forall$x $\in$ S_i | O(s) $\succcurlyeq_i$ O(s [i:x]).

Let G(h)=(N,H|h,P|h, ($\succcurlyeq_i$)|h), a subgame of G starting after history h. A strategy profile s is a *subgame-perfect equilibrium* of G iff for all h $\in$ H, s|h is a Nash Equilibrium of G(h).

Theorem 3.1 (Kuhn)
Every finite extensive game of perfect information has a subgame-perfect equilibrium.

4 The Consensus Problem and Game Theory

This section, presents the consensus problem and represents it as an extensive game with perfect information. A proof of correctness and termination of the algorithm, is achieved by relating these properties with Nash equilibrium in the game.

4.1 The Coordinated Attack Problem

The coordinated attack problem consists of a group of generals which are planning a coordinated attack against a common objective. The only way to succeed is if all generals attack. The generals are located in different places. Nearby generals can only communicate via messengers that travel by foot. However, messenger can be lost or captured. The generals need to agree when they are or not to attack.

The consensus problem is an analogue of the coordinated attack, where a group of processes have to agree in a value of a variable, for instance n=0 or n=1.So, only massage exchange, they must be able to reach a agreement on the value of n. The communication channels are not reliable and massages can be lost.

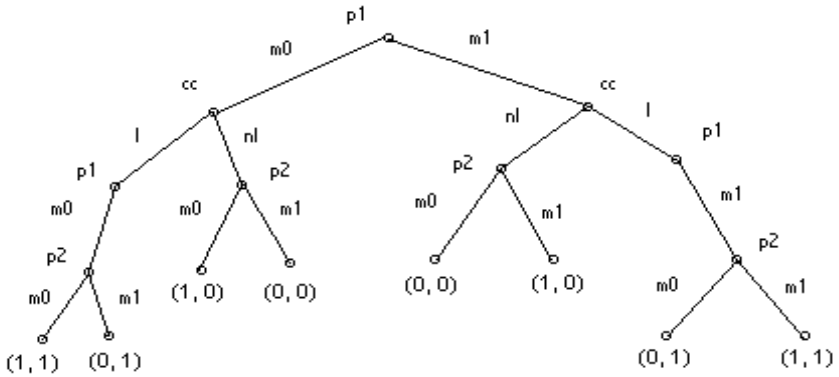
For simplicity, we consider only two processes and the communication channel can lose at most M
messages. All the communications are by broadcast. If a message is not lost it takes T units of time to be delivered and acknowledge, so after T units of time a process knows if its message was delivered or not.

4.2 The Consensus Problem and Extensive Games

The players of this game are the two processes **p1** and **p2** and the third player is the communication channel **cc**. Players p1 and p2 can perform two actions send the value 0 or 1 this action are denoted by **m0** and **m1**. The player cc (communication channel) can also perform two actions: lose the massage currently in the channel or not, denoted respectively by **l** (lose) or **nl** (not lose).

The payoff function is an ordered pair (x, y). It assigns 1 to x when the players agree on the value of n (0 otherwise), and it assigns to y the number of messages lost (for the communication channel). The players p1 and p2 allways get the same payoff, the value of x.

For example, if p1 is the first to play and only one message can be lost, the following game tree is obtained:



This game has two Nash equilibrium, the pair (1,1) in the left hand side and the other one in right hand side.

Theorem 4.1
If M is finite them the consensus problem always reach a consensus in a finite number o message exchange.

Proof: it can be proved by induction in the number of messages lost M, that every history is finite and by Kuhn theorem the game has a subgame-perfect equilibrium, which is the two histories which
End with (1,1,1).

Theorem 4.2
If M is infinite them the consensus problem has no solution, i.e., it can never reach a consensus.

Proof: if M is infinite then the game has infinite histories with the payoff of player cc always increasing, and therefore never reaches an equilibrium.

5 Conclusions

This work models the consensus problem as an extensive game of perfect information and prove correctness and
termination of the algorithm, relating these properties with Nash equilibrium in the game. This open up possibilities of investigating the use of concepts from Game Theory to model distributed algorithms and establish desirable properties.

6 References

[LL78] Lamport, L., 1978, "Time, Clocks, and the Ordering of Events in a Distributed System", *Comm. of the ACM*, 21, pp. 558-565
[NL96] Lynch, N., 1996, "Distributed Algorithms", Morgan Kaufmann, CA, USA.
[OR94] Osborne, M., Rubinstein, A., 1994, "A Course in Game Theory", MIT Press, USA.