Self-Similar Algorithms for Dynamic Distributed Systems*

K. Mani Chandy and Michel Charpentier[†]
Computer Science Department, California Institute of Technology, Mail Stop 256-80
Pasadena, California 91125
mani@cs.caltech.edu, charpov@cs.unh.edu

Abstract

This paper proposes a methodology for designing a class of algorithms for computing functions in dynamic distributed systems in which communication channels and processes may cease functioning temporarily or permanently. Communication and computing may be interrupted by an adversary or by environmental factors such as noise and power loss. The set of processes may be partitioned into subsets that cannot communicate with each other; algorithms in which all such subsets behave in a similar fashion, regardless of size and identities of processes, are called self-similar algorithms. Algorithms adapt to changing conditions, speeding up or slowing down depending on the resources available. The paper presents necessary and sufficient conditions for the application of a self-similar strategy. Self-similar algorithms are developed for several problems by applying the methodology.

1 Introduction

1.1 Contribution

We develop algorithms for dynamic distributed systems in which processes may be repeatedly disabled and then re-enabled during the course of a computation. A disabled process executes no actions and does not change state. Communication between processes may be disrupted temporarily or permanently. We use the terms *agent* and *process* interchangeably because *agent* is used more often than *process* in fields, such as mobile-agent systems, which are relevant to this paper.

Dynamic distributed systems are useful in analyzing several types of problems. In an adversarial situation an opposing team may disable agents and communication channels. In mobile agent systems, agents go in and out of communication range as they travel. Agents may cease functioning after they run out of battery power and resume operation when they gain access to other sources of power. Agents communicating by wireless interfere with each other. This paper discusses algorithms in which agents meet a collective goal—such as computing a function of sensor values—despite the dynamic nature of the system. The algorithms speed up or slow down depending on the resources available. This paper:

- proposes a model for distributed systems composed of sets of agents operating in extremely dynamic and possibly adversarial environments;
- presents a methodology for developing a class of algorithms for dynamic systems;
- identifies necessary and sufficient conditions for the application of the methodology;
- and applies the methodology to example problems.

This paper contributes to research on robust systems by developing multi-agent algorithms that perform correctly under a variety of conditions. The example problems solved in this paper are simple provided the environment is benign. Our challenge is to develop classes of algorithms that perform efficiently when conditions permit, and perform correctly even under adverse conditions

1.2 Agents and their Environment

A system consists of a set of agents and an environment in which the agents operate. The environment has a state space, and each agent has a state space. The state of the environment determines how agents may interact with each other. A set of agents may be able to communicate with each other in one state of the environment and unable to communicate in a different environmental state. An agent may be disabled—unable to change state—in one state of the environment and enabled in another environmental state.

A set of communicating agents can execute collaborative algorithms that change the states of participating agents.

*This work is supported by AFOSR MURI award FA9550-06-1-0303.

[†] On leave from the University of New Hampshire, 2006/07

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE

When the environment does not permit these agents to communicate or compute they cannot execute collaborative algorithms. We model the environment's ability to enable and disable agents and communication channels by specifying the agent transitions that are enabled in each state of the environment. Agent transitions may become enabled or disabled as the environment's state changes.

The environment does not change the states of agents though it can disable agent transitions. Agents do not change the state of the environment nor enable or disable transitions of the environment's state. Designers cannot specify the environment's states and transitions; these are given in the problem specification. Designers can, however, specify certain progress properties assumed of the environment. If no restrictions are placed on the environment then the environment can permanently disable all agents. The designer's challenge is to find weak constraints on the environment that guarantee that the agents reach their goals eventually.

What should agents in a group do when they cannot communicate with agents outside the group? Since all the agents outside the group may be disabled, the group behaves as though the entire system consisted of that group of agents alone. We give the name *self-similar computations* for computations in which each group of communicating agents acts as though the total system consisted of that group alone. A definition of self-similarity in the literature is that a self similar object is similar to part of itself. Likewise, a self-similar computation by a group of agents is similar to computations by its subgroups of agents. Self-similar algorithms can be defined succinctly because a description is given for a single algorithm that is employed by all groups regardless of size and identity.

The state spaces of agents may be discrete or continuous. We have started to study algorithms for systems in which variables change value continuously with time, and in which dynamics are specified by differential or difference equations. This paper restricts attention to discrete state transition systems that are specified using temporal logic.

Our model is defined in sect. 2. The methodology and the conditions of its applications are presented in sect. 3. Examples are developed in sect. 4. Section 5 discusses related work and summarizes the paper's contributions. Most proofs are omitted or are presented in outline form, due to insufficient space; detailed proofs are given in [2].

2 A Model of Dynamic Distributed Systems

2.1 Dynamic Systems as Transition Systems

Transition Systems A (fair) transition system is defined by (*i*) a set of states, (*ii*) a predicate on states that specifies the set of initial states, (*iii*) a relation → between pairs of states, and (*iv*) a progress property. A computation is a sequence of states, starting from an initial state, where $X \rightarrow X'$ holds for each successive pair of states X and X' and the computation satisfies the progress property.

States of Dynamic Distributed Systems A dynamic distributed system has a fixed set $\mathcal{A}$ of agents. Each agent a has a state space. A system state is a pair $(\mathbf{G}, \mathbf{S})$ where $\mathbf{G}$ is the state of the environment and $\mathbf{S}$ is the multiset (or bag) of states of agents in the system. We use G, G' and S, S' to denote values of $\mathbf{G}$ and $\mathbf{S}$, respectively. We use bold symbols such as $\boldsymbol{\in}$, $\boldsymbol{\cup}$, $\boldsymbol{\{}$ or $\boldsymbol{\}}$ to represent multiset operations and ordinary symbols such as $\in$, $\cup$, $\{$ or $\}$ to represent (ordinary) set operations. For any system state (G, S) and agent a, let S_a be the state of agent a while the system is in state (G, S); then: $S = \boldsymbol{\{} S_a \mid a \in \mathcal{A} \boldsymbol{\}}$.

State Transitions of Dynamic Distributed Systems Let B be a subset of the agents. Let S_B be the multiset of states of agents in B when the system is in state (G, S): $S_B = \boldsymbol{\{} S_a \mid a \in B \boldsymbol{\}}$. As a consequence, $S = S_{\mathcal{A}}$ and, for any disjoint subsets B and C of $\mathcal{A}$, $S_{B \cup C} = S_B \boldsymbol{\cup} S_C$.

A set B of agents can execute a collaborative algorithm that changes S_B. The collaborative algorithm executed is described by a relation $\mathcal{R}$ that specifies the state transitions of the set B from S_B to S'_B while the environment is in state G:

$$(G, S_B)\ \mathcal{R}\ (G, S'_B)$$

We require that relation $\mathcal{R}$ is reflexive—for any set B of agents and any state G of the environment, $(G, S_B)\ \mathcal{R}$ (G, S_B)—because, under any environment, a group of agents can always leave the state of the system unchanged. In the following, the phrase: "transition of B from S_B to S'_B" means that S_B and S'_B are states of a group B of agents where $(G, S_B)\ \mathcal{R}\ (G, S'_B)$ holds for some G.

The entire state $(\mathbf{G}, \mathbf{S})$ can change because of a change in either the environment $\mathbf{G}$ or the agents $\mathbf{S}$ but not by both simultaneously. Furthermore, disjoint sets of agents can execute the algorithm concurrently. The possible state changes of $\mathbf{S}$ are specified by $\mathcal{R}$. Our goal is to develop algorithms that work for all types of environments. Therefore, we place no direct constraints on state transitions of the environment. We do, however, place constraints—specified as progress properties—on infinite computations.

Let π be a partition of the set A of agents into groups. Each group B in π can make a state transition from S_B to S'_B provided $(G, S_B)\ \mathcal{R}\ (G, S'_B)$ holds. A state transition from S to S' of the set of all agents is one in which every group B of agents in π makes a state transition from S_B to S'_B (including the possibility $S'_B = S_B$) and hence $\forall B \in \pi : (G, S_B)\ \mathcal{R}\ (G, S'_B)$. Since no constraints are placed on transitions of the environment's state, the environment may transit from a state G to *any* state G'. A transition of the entire system is either a transition of the environment from state G to any G' while agent states remain unchanged, or a transition of the states of agents from S to S' while the environment state remains unchanged. The transition relation for the entire system $(\mathbf{G}, \mathbf{S})$ is defined as follows:

$$\begin{aligned}((G,S) \rightarrow (G',S')) \equiv \\ (S = S') \;\vee \\ ((G = G') \;\wedge\; (\exists \pi : \forall B \in \pi : (G, S_B) \,\mathcal{R}\, (G, S'_B)))\end{aligned}$$

where π is a partition of the set $\mathcal{A}$ of agents.

Progress Properties of Dynamic Distributed Systems No progress is possible while the environment prevents all agents from changing state. We propose a model of dynamic systems that makes no assumptions about transitions of the environment's state over finite computations and makes only weak assumptions about environment state transitions over infinite computations. A key issue is to identify requirements about the environment that guarantee that agents do not remain stuck in the same state forever. We define a relation $\looparrowright$ between the state of the environment and the state of the agents where $S \looparrowright G$ means that while the environment is in state G the agents can transit from S to a different state.

$$S \looparrowright G \;\equiv\; (\exists S' : S' \neq S : (G,S) \rightarrow (G,S'))$$

Read $S \looparrowright G$ as "S escapes G". Note that "agents can escape S when the environment is in G" means only that there exists a transition of agents from state S to some other state S' while the environment is in state G; no assumptions are made about S' other than that it is different from S.

Let Q be a predicate on states G of the environment. We extend the definition of relation $\looparrowright$ from environmental states to predicates on environmental states: $S \looparrowright Q$ is true when the environment allows a transition of agent states from S to a different state while Q holds.

$$S \looparrowright Q \;\equiv\; (\forall G : Q(G) : S \looparrowright G)$$

If agents can escape S when the environment is in any state G that satisfies Q, and the environment satisfies Q infinitely often, then agents in S either transit to a different state or are given the opportunity to do so infinitely often. We restrict our attention to dynamic systems that satisfy the following requirement, which we call the *escape postulate*. **Escape Postulate**: If agents can transit from a state infinitely often then they will do so eventually.

$$\forall S : S \looparrowright Q : \Box\Diamond Q \implies \Box\Diamond(\mathbf{S} \neq S) \qquad (1)$$

The operators $\Box$ and $\Diamond$ are the usual *henceforth* and *eventually* of linear-time temporal logic [6]: $\Box\Diamond P$ means that, in any computation of the system, predicate P holds infinitely often. The escape postulate is not too restrictive; nevertheless we need to prove that it holds for a given implementation. One can, for example, imagine the following system for which the escape postulate does not hold: the environment always transits from G to G' before the agents can take a step, where both G and G' satisfy Q. In this case, even though agents can escape S while the environment is in G or G' and though Q holds infinitely often, the agents may remain stuck in S forever.

2.2 Specification of Algorithms

The specification of an algorithm for a dynamic distributed system consists of a specification of agents and a specification of the environment.

Specification of the Algorithm of Agents The specification of agents consists of a specification of their state spaces, their initial values and the state-transition relation $\mathcal{R}$.

Specification of the Environment The state space, initial states, and transition relations of the environment are given and designers cannot specify these values. Designers may specify a set $\mathcal{Q}$ of predicates on environmental states and require that every predicate in $\mathcal{Q}$ holds infinitely often:

$$(\forall Q \in \mathcal{Q} : \Box\Diamond Q) \qquad (2)$$

The environment may change its state in any arbitrary way subject to this constraint.

2.3 Discussion of the Model

Different models of dynamic systems are suitable for different purposes. In this paper we propose a simple model in which (*i*) agents are specified by a *single* state-transition relation $\mathcal{R}$, (*ii*) there are no assumptions about the environment regarding finite computations, and (*iii*) assumptions about infinite computations are defined by a set $\mathcal{Q}$ of predicates where (2) holds.

An alternate approach is to specify agents as action systems with fairness criteria about action execution. Action system models are more descriptive than models defined by a single state-transition relation in the sense that more items—actions, fairness criteria—are used in specifying the model. For some problems we can prove more properties about action-system models than the single-relation model; however, we chose to begin with the single-relation model for reasons of simplicity.

A challenge in designing algorithms is to find a set $\mathcal{Q}$ such that (2) is a weak restriction on the environment. We do not define "weak" and "strong" restrictions on the environment formally because their meanings are problem dependent. For instance, in many applications agents are distributed in space, and to require that agents near each other can communicate with each other is a weaker restriction on the environment than to require that agents far away from each other can communicate with each other. We give a few examples of suitable sets $\mathcal{Q}$ in section 4 where several examples are considered.

In the remainder of the paper, we restrict our attention to a specific set of problems that occur in dynamic distributed systems. The state spaces and initial states of agents are given for these problems and the objective is for agents to compute a function of the initial state. Our design task is to specify $\mathcal{R}$ and $\mathcal{Q}$ where we can prove that the algorithm defined by $\mathcal{R}$ satisfies the specification provided assumption (2) on $\mathcal{Q}$ holds. The states, transitions and initial state of the environment are not specified—the only design parameter regarding the environment is $\mathcal{Q}$.

3 A Class of Algorithms for Dynamic Systems

3.1 Problem Specification

We are given a function f from agent states to agent states where f is idempotent: $f(f(S)) = f(S)$. We are required to design a system such that if the agents state $\mathbf{S}$ has an initial value $S^{(0)}$ then within a finite number of steps a point in the computation is reached after which $\mathbf{S} = f(S^{(0)})$ forever:

$$(\mathbf{S} = S^{(0)}) \implies \Diamond\Box(\mathbf{S} = f(S^{(0)})) \tag{3}$$

In a sensor network, for instance, f is a function on agent variables such as values read by sensors. If f computes the average of sensor values then the specification requires that in a finite number of steps $\mathbf{S}$ becomes and remains the average of the initial values $S^{(0)}$ provided (2) is satisfied.

3.2 Consequences of the Specification

Terminology We use the following concepts from temporal logic [6]: $\leadsto$ ("leads-to") and stable . Their meanings are as follows. "$P \leadsto Q$" means if P is true at any point in a computation then Q is true at that point or a later point: $(P \leadsto Q) \equiv \Box(P \implies \Diamond Q)$. The property "$\mathsf{stable}\ P$" means if P holds at any point in a computation then it continues to hold thereafter: $(\mathsf{stable}\ P) \equiv \Box(P \implies \Box P)$

Lemma If the agents are in state S at any point in a computation then the agents will eventually remain forever in state $f(S)$ in that computation:

$$\Box((\mathbf{S} = S) \implies \Diamond\Box(\mathbf{S} = f(S)))$$

Proof: Suppose the system reaches a state $\mathbf{S} = S$. Because $\mathbf{S}$ represents the *entire* state of the multi-agent system, there is nothing that distinguishes the remainder of this computation from a computation that begins with the initial state $\mathbf{S} = S$. From (3) such a computation must eventually reach a state where $\mathbf{S}$ becomes and remains equal to $f(S)$.

Theorem: Conservation Law An invariant of programs that satisfy the specification (3) is $f(\mathbf{S}) = S^*$ where $S^* = f(S^{(0)})$. (We call this property the conservation law because $f(\mathbf{S})$ is conserved.)

$$\Box(f(\mathbf{S}) = S^*)$$

Proof: Let S and S' be states of agents in a computation. From the previous lemma there is an infinite suffix (tail) in which $\mathbf{S} = f(S)$ and $\mathbf{S} = f(S')$. Hence $f(S) = f(S')$. Substituting $S^{(0)}$ for S' gives that $f(S) = S^*$ for all reachable states S.

Lemma: Stability

$$\mathsf{stable}\ (\mathbf{S} = f(S))$$

Proof: Suppose there exists a transition from $f(S)$ to a different state S' in some state G of the environment. Whenever the state $f(S)$ is reached in the future, the environment could be in state G, thus enabling the transition again. Therefore, there exists a possible computation in which $f(S)$ is always followed by S'. This contradicts the previous lemma that the system must eventually reach and remain in a state $\mathbf{S} = f(f(S)) = f(S)$.

Alternate Specification From the above discussion, an alternate specification is to design an algorithm in which $\mathbf{S} = f(S)$ is stable and is eventually reached. Therefore, the original specification (3) can be replaced with:

$$\mathsf{stable}\ (\mathbf{S} = f(S)) \tag{4}$$

$$(\mathbf{S} = S) \leadsto (\mathbf{S} = f(S)) \tag{5}$$

3.3 Consequences of Self Similarity

From the conservation law, any state transition conserves $f(\mathbf{S})$. We design algorithms in which transitions by all groups of agents are self-similar. In particular, any group B behaves like the entire system and hence preserves f *for that group*.

Group Conservation Law Every state transition from S_B to S'_B of a group B preserves f for that group:

$$f(S_B) = f(S'_B)$$

Suppose the set of agents is partitioned into groups B and C, both of which take concurrent steps that preserve f for their groups. These steps constitute a step of group $B \cup C$ and therefore must preserve f for that group. We restrict attention to functions f for which the following property holds:

Local Conservation Implies Global Conservation

$$(f(S_B) = f(S'_B)) \wedge (f(S_C) = f(S'_C)) \wedge (B \cap C = \varnothing) \Rightarrow (f(S_{B\cup C}) = f(S'_{B\cup C}))$$

3.4 Super-Idempotent Functions

A function f from (finite) multisets to multisets is defined to be *super-idempotent* if and only if it satisfies the following equation for any multisets X and Y:

$$f(X \cup Y) \;=\; f(f(X) \cup Y)$$

A super-idempotent function is also idempotent as is evident by setting Y to the empty set in the above equation. The next theorem states that super-idempotent functions are exactly those idempotent functions for which the "local conservation implies global conservation" property holds.

Theorem: Local Conservation Implies Global Conservation Exactly for Super-Idempotent Functions A function f is super-idempotent if and only if it is idempotent and satisfies the following property.
For any multisets X, X', Y and Y':

$$(f(X) = f(X')) \wedge (f(Y) = f(Y')) \implies (f(X \cup Y) = f(X' \cup Y'))$$

The following theorem allows for simpler checks to determine if a function is super-idempotent.

Theorem: A function f from multisets to multisets is super-idempotent if and only if it is idempotent and it satisfies the following property. For any multiset X and value v:

$$f(X \cup \{v\}) = f(f(X) \cup \{v\}) \tag{6}$$

The next lemma identifies a sufficient condition for a function to be idempotent. This condition is easily checked for some problems (see section 4 for examples). Let $\circ$ be a binary, associative, commutative operator on multisets. Let X be a nonempty multiset consisting of elements x_j for $0 \le j \le J$ and define $\circ X$ as $\circ X = \{x_0\}\circ\{x_1\}\circ\ldots\circ\{x_J\}$.

Lemma: A sufficient condition for a function f from multisets to multisets to be super-idempotent is that there exists a binary, associative, commutative operator on multisets such that: $f(\varnothing) = \varnothing$ and $f(X) = \circ X$ for nonempty X.

3.5 Variant Functions

We use a variant function h over the state $\mathbf{S}$, where the range of the function is a well-founded set for some order $>$. We design algorithms where each state change reduces the value of the variant function. From (5), the goal state is S^* from any state S, and hence we propose that the minimum value of h subject to the invariant $f(\mathbf{S}) = S^*$ is taken when $\mathbf{S} = S^*$:

$$(f(S) = S^*) \wedge (S \neq S^*) \implies h(S) > h(S^*)$$

Next we present results for h that follow in exactly the same way as the corresponding results for f. We call a state change *an improvement* exactly when it reduces the value of h. Since our algorithms are self-similar we design them with the following property:

Group State Changes are Improvements For any transition of a group B from S_B to S'_B:

$$S_B \neq S'_B \implies h(S'_B) < h(S_B)$$

Local Improvement Implies Global Improvement For any disjoint sets B and C of agents, and any state transitions from S_B to S'_B and from S_C to S'_C that conserve f for both groups (i.e., $f(S'_B) = f(S_B)$ and $f(S'_C) = f(S_C)$):

$$\begin{aligned}&(h(S'_B) < h(S_B)) \wedge (S'_C = S_C)\\ &\qquad\implies (h(S'_{B\cup C}) < h(S_{B\cup C}))\\ &(h(S'_B) < h(S_B)) \wedge (h(S'_C) < h(S_C))\\ &\qquad\implies (h(S'_{B\cup C}) < h(S_{B\cup C}))\end{aligned} \tag{7}$$

Theorem: Function h satisfies (7) if function f is super-idempotent and for all multisets X and X', and element v where $f(X') = f(X)$:

$$(h(X') < h(X)) \implies (h(X' \cup \{v\}) < h(X \cup \{v\}))$$

In many cases, the desired properties of function h can be verified without having to resort to the previous theorem by using the following lemma:

Lemma: Given a super-idempotent function f, a sufficient condition for h to satisfy (7) is that it has the following form, for some family of functions h_a:

$$h(S_B) = \sum_{a\in B} h_a(S_a) \tag{8}$$

3.6 Casting the Given Problem as Constrained Optimization

The previous section shows that we design $\mathcal{R}$ as a constrained optimization algorithm, using the constraint that f is preserved and some objective function h. Therefore, we expect $\mathcal{R}$ to be a refinement of the relation $\trianglerighteq$ defined as follows:

$$S_B \triangleright S'_B \;\equiv\; (f(S_B) = f(S'_B)) \wedge (h(S_B) > h(S'_B))$$
$$S_B \trianglerighteq S'_B \;\equiv\; (S_B \triangleright S'_B) \vee (S_B = S'_B)$$

The relation $\trianglerighteq$ expresses that groups of agents take optimization steps in which f is conserved and h decreases for the group.

3.7 Distributed Algorithms for Constrained Optimization

To verify the correctness of an algorithm defined in terms of $\trianglerighteq$, designers must discharge three proof obligations. The proof obligations are as follows, where S and S' are universally quantified over all the reachable states of the system:

Proof Obligation: $\mathcal{R}$ Implements $\trianglerighteq$

$$((G, S)\,\mathcal{R}\,(G, S')) \implies (S \trianglerighteq S')$$

This first proof obligation expresses that every step taken by the algorithm $\mathcal{R}$ is a valid optimization step.

Proof Obligation: Agents Eventually Transit out of Nonoptimal States

$$(S \neq S^*) \implies (\exists Q \in \mathcal{Q} : S \looparrowright Q) \qquad (9)$$

This second proof obligation expresses that nonoptimal states can be escaped when each predicate in $\mathcal{Q}$ is enabled infinitely often. Note that this proof obligation deals with both the environment *and* $\mathcal{R}$. Once proved, this formula can be combined with the assumption on the environment (2) and the escape postulate (1) to show that agents eventually transit from any nonoptimal state.

Proof Obligation: Local-to-Global Property

$$(S_B{\trianglerighteq}S'_B){\wedge}(S_C{\trianglerighteq}S'_C){\wedge}(B{\cap}C = \varnothing) \implies (S_{B\cup C}{\trianglerighteq}S'_{B\cup C}) \qquad (10)$$

The last proof obligation expresses that if two groups of agents take optimization steps concurrently, their set union changes state in a way that is compatible with $\trianglerighteq$.

Theorem (Correctness): An algorithm specified by a transition relation $\mathcal{R}$ and a set $\mathcal{Q}$ such that they satisfy (3.7), (9) and (10) solves the problem of computing $f(S^{(0)})$ according to specifications (4) and (5).

4 Systematic Derivation of Algorithms

In this section we develop algorithms for several problems using our method. We illustrate consensus and nonconsensus examples, as well as examples for which the given function f is not super-idempotent. For each example, we propose a suitable function h that has the summation form of (8). Therefore, the resulting constrained optimization step $\trianglerighteq$ satisfies the local-to-global proof obligation as long as the function f is super-idempotent.

We present one or more possible sets $\mathcal{Q}$ to constrain the environment. These sets are defined in terms of a graph in the following way. Consider a graph $(\mathcal{A}, E)$ whose vertices are agents and edges are communication links. Consider an edge e from E and define a predicate Q_e to mean that the edge e exists and is available for communication. Define $\mathcal{Q}_E = \{Q_e \mid e \in E\}$.

We do not describe explicit refinements $\mathcal{R}$ of the relation $\trianglerighteq$. Such refinements depend on specific assumptions on the environment of the system (e.g., what type of communication is available to agents) and are beyond the scope of this paper. When it is informative, we briefly describe possible strategies for implementing $\trianglerighteq$ through $\mathcal{R}$.

4.1 Minimum of a Set

As a first example, consider the task of computing the minimum of a distributed set of values as a consensus problem: The desired final state is one in which every agent has computed the minimum.

Agents State: Each agent a has a single integer variable x_a, initially equal to $x_a^{(0)}$. We assume that $\forall a : x_a^{(0)} \geq 0$.

Distributed Function: The task is to compute $f(S^{(0)})$ where $S^{(0)} = \{x_a^{(0)} \mid a \in \mathcal{A}\}$ and f of a multiset is a multiset of same cardinality in which all the values are equal to the minimum of the original multiset. For instance, $f(\{3, 5, 3, 7\}) = \{3, 3, 3, 3\}$. Note that $f(X)$ is of the form $\circ X$ for a commutative associative operator $\circ$ and hence is super-idempotent.

Objective Function: We define the objective function h as follows:

$$h(\mathbf{S}) = \sum_{a \in \mathcal{A}} x_a$$

Function h is integer valued and only takes nonnegative values, so its range is well-founded.

Algorithm $\mathcal{R}$: The relation $\mathcal{R}$ can be any algorithm where each state change maintains the minimum of a group while reducing the sum of that group. For instance, all the agents

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE

in a group may update their value to any value between their current value and the minimum of the group.

Assumption $\mathcal{Q}$ on the Environment: It is straightforward to verify that, for any connected graph E, the set $\mathcal{Q}_E$ satisfies the proof obligation (9).

4.2 Sum of a Set

Consider now a similar problem where the task is to compute the sum of values held by individual agents. Unlike the minimum, this problem cannot be solved as a simple consensus, because if each agent replaces its value with the global sum, this sum changes. In other words, as a consensus, the function f that defines the sum problem is not idempotent. Instead, we solve a different problem in which the requirement is for *one* agent to obtain the sum while all the other agents have value zero.

Agents State: Each agent a has a single integer variable x_a, initially equal to $x_a^{(0)}$. We assume that $\forall a : x_a^{(0)} \geq 0$.

Distributed Function: The function f transforms a multiset into a multiset with only two distinct values: the sum of the numbers from the original multiset (with multiplicity 1) and zero (with multiplicity $N-1$). For instance, $f(\{3,5,3,7\}) = \{18,0,0,0\}$. Function f is defined by a commutative associative operator and is super-idempotent.

Objective Function: We choose a function h defined as:

$$h(\mathbf{S}) = \left(\sum_{a\in\mathcal{A}} x_a\right)^2 - \sum_{a\in\mathcal{A}} x_a^2$$

As in the previous example, function h is nonnegative and integer-valued and hence its range is well-founded.

Algorithm $\mathcal{R}$: The relation $\mathcal{R}$ defines state changes that maintain the sum of a group while increasing the squares of the values of that group. This can be achieved by having values move away from each other, for instance by making small values smaller and large values larger.

Assumption $\mathcal{Q}$ on the Environment: In this example, it is important that the agent that will eventually compute the sum keeps interacting with other non-zero agents *directly*. Zero agents do not have any meaningful interaction and cannot be used as intermediates between non-zero agents. Since the assumption $\mathcal{Q}$ that we can make on the environment is independent from the values of agents, the weakest assumption that guarantees termination is that any two agents have the opportunity to communicate infinitely often. This corresponds to a $\mathcal{Q}_E$ in which E is a *complete* graph.

4.3 Second Smallest Value

This example is a variation of the minimum example. This time, the problem is to compute the second smallest value of a set. We define the second smallest value of a multiset as the smallest value that is different from the minimum, if such a value exists. When all the values in a multiset are equal, we define the second smallest to be this value.

Agents State: Each agent a has a single integer variable x_a, initially equal to $x_a^{(0)}$.

Distributed Function: The task is to compute $f(S^{(0)})$ where $S^{(0)} = \{x_a^{(0)} \mid a \in \mathcal{A}\}$ and f of a multiset is a multiset of same cardinality in which all the values are equal to the second smallest value of the original multiset, as defined above. For instance, $f(\{3,5,3,7\}) = \{5,5,5,5\}$.

Function f is idempotent but not super-idempotent: Let $X{=}\{1,3\}$ and $Y{=}\{2\}$; $f(X) = \{3,3\}$ and $f(f(X) \cup Y) = f(\{3,3,2\}) = \{3,3,3\}$, but $f(X \cup Y) = f(\{1,3,2\}) = \{2,2,2\}$.

Since f is not super-idempotent, optimization steps defined by the relation $\trianglerighteq$ cannot satisfy the local-to-global proof obligation (10). Therefore, the self-similar approach we advocate cannot be applied directly to this problem. It is possible, however, to generalize the given problem to one for which the self-similar approach works: compute *both* the smallest and second smallest values.

Agents State: Each agent a now has a pair of variables (x_a, y_a), initially equal to $(x_a^{(0)}, x_a^{(0)})$ (i.e., the first and second members of each pair are identical). We assume that $x_a^{(0)} \geq 0$, for all agents a.

Distributed Function: The task is to compute $f(S^{(0)})$ where $S^{(0)} = \{(x_a^{(0)}, x_a^{(0)}) \mid a \in \mathcal{A}\}$ and f of a multiset is a multiset of same cardinality in which all the values are equal to the pair (x,y), where x and y are the smallest two of all the values that appear in the original multiset, as a first or second element of a pair, except when all the values of the original multiset are equal, in which case the multiset is unchanged. For instance, $f(\{(2,5),(3,4),(2,7)\}) = \{(2,3),(2,3),(2,3)\}$, $f(\{(2,2),(2,2)\}) = \{(2,2),(2,2)\}$.

Function f is clearly idempotent. To show that it is super-idempotent, one needs to verify that it satisfies (6), for any multiset X and any pair (x,y) (see [2] for details).

Objective Function: The objective function h is defined as a generalization of the summation function used in the minimum example:

$$h(\mathbf{S}) = \sum_{a\in\mathcal{A}} (x_a + y_a)$$

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE

This example shows how one can deal with an idempotent function that is not super-idempotent by transforming the original problem into a suitable (super-idempotent) problem. The strategy is illustrated again in section 4.5 on a different example. An obvious downside is that the transformed problem requires agents to have access to a larger memory (two values instead of one), a drawback that will be even worse if one is interested in computing the k-th smallest value of set. An alternative to computing the k-th smallest value this way is to sort the entire set of values in place (no additional memory required). We study the problem of sorting a set of values in the following section.

4.4 Sorting

The problem is to sort an array of numbers in nondecreasing order, where each agent holds one value from the array. The proposed solution can easily be generalized to the case where each agent holds one or more contiguous ranges of the array instead of a single value.

Agents State: Each agent a holds a pair of values (i_a, x_a), which represent an index and a corresponding value in the distributed array. We assume that the set of indexes is totally ordered for some order relation $<$ and that the set of values is totally ordered for some order relation $\prec$, possibly distinct from $<$. We also assume that all the i_a are distinct.

Distributed Function: The task is to compute $f(S^{(0)})$ where $S^{(0)} = \{\!\{(i_a^{(0)}, x_a^{(0)}) \mid a \in \mathcal{A}\}\!\}$ and f of a multiset X is the unique multiset Y of same cardinality with the following properties:

$$(\forall (i_a, x_a), (i_b, x_b) \in Y : i_a < i_b \implies x_a \preceq x_b)$$
$$\{\!\{i \mid (i,x) \in X\}\!\} = \{\!\{i \mid (i,x) \in Y\}\!\}$$
$$\{\!\{x \mid (i,x) \in X\}\!\} = \{\!\{x \mid (i,x) \in Y\}\!\}$$

For example, $f(\{\!\{(1,3),(2,5),(3,3),(4,7)\}\!\})$ equals $\{\!\{(1,3),(2,3),(3,5),(4,7)\}\!\}$. This function is clearly idempotent (sorting a sorted array has no effect). To see that function f is also super-idempotent, observe that $f(X)$ differs from X by a permutation of the values w.r.t. the indexes. To sort an array directly of after some values have been permuted yields the same sorted array.

Objective Function: A classic objective function for sorting problems is a function that counts the number of out-of-order pairs in the array:

$$h(\mathbf{S}) = |\{(a,b) \in \mathcal{A} \times \mathcal{A} \mid (i_a < i_b) \wedge (x_b \prec x_a)\}|$$

The range of h is well founded. However, this objective does not satisfy the desired local-to-global property (10) of relation $\trianglerighteq$, as shown on the following counterexample.

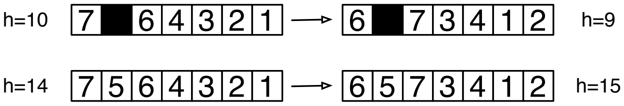

Figure 1. "Number of out-of-order pairs" does not have the local-to-global property.

Suppose that $\mathcal{A}$ consists of 7 agents and that the set of indexes and the set of values are both equal to the set of integers $\{1 \cdots 7\}$. We represent the state $\mathbf{S}$ of the system as the list of values held by agents, in order of their indexes. Consider the state $S = [7,5,6,4,3,2,1]$; this is a state in which agent 1 holds value 7, agent 2 holds value 5, etc. Let $\mathcal{A}$ be partitioned into two groups $B = \{1,3,4,5,6,7\}$ and $C = \{2\}$ and assume a transition from S to $S' = [6,5,7,3,4,1,2]$. We have $f(S'_B) = f(S_B)$ and $S'_C = S_C$. One can also see that $h(S_B) = h([7,6,4,3,2,1]) = 10$ and $h(S'_B) = h([6,7,3,4,1,2]) = 9$. Thus, the pair (S, S') satisfies $S_B \trianglerighteq S'_B$ and $S_C \trianglerighteq S'_C$. However, we can see that $h(S_{B\cup C}) = h([7,5,6,4,3,2,1]) = 14$ and $h(S'_{B\cup C}) = h([6,5,7,3,4,1,2]) = 15$. Therefore, $S_{B\cup C} \trianglerighteq S'_{B\cup C}$ is not satisfied and (10) does not hold. Fig. 1 describes the transition from S_B to S'_B (for which h decreases) and the corresponding transition from $S_{B\cup C}$ to $S'_{B\cup C}$ (for which h increases).

It is important to note that the situation here is different from the second smallest value example, in which (10) was not satisfied because f was not super-idempotent. Since f here *is* super-idempotent, a different objective function h might be chosen to make relation $\trianglerighteq$ satisfy the local-to-global property. We propose to replace the previous function with a new objective function. To simplify the definition, assume the indexes i_a are a consecutive range of integers and the values $x_a^{(0)}$ to be sorted are distinct. Then, each value x_a can be associated with a unique index $\mathsf{ord}(x_a)$ that is the desired position of x_a in the sorted array. The objective function is defined as follows:

$$h(\mathbf{S}) = \sum_{a \in \mathcal{A}} (i_a - \mathsf{ord}(x_a))^2$$

Informally, h is the sum of the squares of the distances between the current position and the desired position in the array, for all values. This new objective function has the summation form of (8) and its range is well-founded.

Algorithm $\mathcal{R}$: The relation $\mathcal{R}$ can be any algorithm that permutes the elements of a group while reducing h. It can be verified, from the definition of h, that any swap of one or more out-of-order pairs of elements decreases the value of the function. Therefore, any sorting algorithm that works by swapping out-of-order pairs is acceptable.

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE

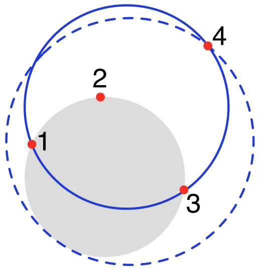

Figure 2. "Circumscribing function" not super-idempotent.

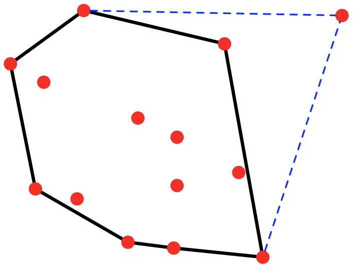

Figure 3. "Convex hull function" is super-idempotent.

Assumption $\mathcal{Q}$ on the Environment: Although this example is, like the sum, an example of a non-consensus algorithm, it does not require the environment to enable the edges of a complete graph. It is enough that each agent can communicate infinitely often with the agents corresponding to the positions on the left and on the right of its own index in the array. Therefore, a possible constraint on the environment is a set Q_E in which E is a linear graph of all the agents in order of their indexes.

4.5 Circumscribing Circle

As a last example, we consider a consensus problem with a geometric flavor. In this example, each agent represents a point in a two-dimensional space, and agents need to compute the circumscribing circle of all these points, i.e., the unique circle of smallest area such that all the points are on or inside the circle.

Agents State: Each agent a maintains its (constant) coordinates, as well as its own estimate of the circumscribing circle, as a center point and radius. So, the state of an agent is a 5-tuple (X_a, Y_a, x, y, r), where (X_a, Y_a) represent the coordinates of the agent and (x, y, r) is the agent's current view of the circumscribing circle. Initially, $x^{(0)} = X_a$, $y^{(0)} = Y_a$ and $r^{(0)} = 0$.

Distributed Function: Informally, the function f to be computed is defined as follows. Given a multiset of agent states, it builds a multiset of same size in which the (x, y, r) part of each 5-tuple is modified so they are all equal to (X, Y, R), where (X, Y, R) defines the smallest circle that contains all the circles (x, y, r) of the agents in the multiset. It follows immediately that $f(S^{(0)})$ is the circumscribing circle of the points represented by the agents. It is also clear that f is idempotent. However, as shown on fig. 2, the function f is not super-idempotent.

Assume B is the set that consists of agents $1, 2$ and 3 and C is the singleton $\{4\}$, agents $1, 2$ and 3 have computed the exact circumscribing circle for these three agents, and the state of agent 4 is a point (circle of radius zero) centered on (X_4, Y_4). In the case depicted in fig. 2, $f(S_B \cup S_C)$ (solid line circle) and $f(f(S_B) \cup S_C)$ (dashed line circle) are different, which means that f is not super-idempotent.

As in the example of the second smallest value, the given problem needs to be transformed into a more general problem for which the computed function is super-idempotent and a suitable objective function h can be found. In this case, a possible generalized problem is to compute the convex hull of a given set of points, from which the circumscribing circle can easily be obtained.

Fig. 3 shows a set of points and their convex hull (on the left) and one additional point (on the right). A geometrical argument can be used to show that the convex hull of all the points is equal to the convex hull of all the points of the original convex hull plus the additional point. From this, it follows that the convex hull function is super-idempotent.

Agents State: Each agent maintains, in addition to its (constant) coordinates (X_a, Y_a), a set of points V_a that represents its current hull. Initially, $V_a^{(0)} = \{(X_a^{(0)}, Y_a^{(0)})\}$.

Distributed Function: The function f transforms a multiset of agent states into a multiset of same size in which all variables V_a are equal to the convex hull of the union of all the points in the V_a sets of the original multiset. This function is super-idempotent.

Objective Function: A possible objective function h is defined in terms of a constant P and a function $\mathsf{perimeter}$:

$$h(\mathbf{S}) = |\mathcal{A}|.P - \sum_{a \in \mathcal{A}} \mathsf{perimeter}(V_a)$$

P is the perimeter of the global convex hull to be computed and $\mathsf{perimeter}(V_a)$ is the perimeter of the convex hull defined by V_a. The range of h is a finite set (the perimeters of the convex hulls of all the subsets of points) and therefore is well-founded for $<$.

Algorithm $\mathcal{R}$: A possible strategy to define relation $\mathcal{R}$ is for groups to compute convex hulls that include two or more of the current hulls as already computed by agents. A group of agents (a_i) each with a convex hull V_{a_i} can replace their hull with the convex hull of the points in $\bigcup_i V_i$. A geometrical argument shows that such a step increases the sum of the perimeters of the hulls, hence that h decreases. There are however more possible strategies. In particular, it is not required that all the agents in a group compute the same hull at each step; so $\mathcal{R}$ can be easily implemented by asynchronous message passing: an agent a can update V_a upon receiving a message without requiring that the sender of the message changes its own estimate of the hull.

Assumption $\mathcal{Q}$ on the Environment: A suitable choice for parameter $\mathcal{Q}$ is $\mathcal{Q}_E$ where E is any connected graph.

5 Related Work and Conclusions

A methodology for solving the problems discussed in our paper is for each agent to take repeated global snapshots or to employ group communication protocols—see the survey [3]; these approaches work well in systems that are relatively static but are inefficient in dynamic systems.

Fault tolerance in dynamic distributed systems is studied in [11, 7, 9]. Pioneering studies and surveys of consensus algorithms include [5, 8]. Decentralized iterative schemes for consensus problems have been proposed by [4, 12]. Distributed algorithms based on convex optimization are presented in [1]. Consensus problems in networks of agents with switching topology are treated in [10].

This paper builds upon earlier work by: (*i*) dealing with a class of problems that includes, but is not restricted to, consensus among agents; (*ii*) developing *classes of algorithms* for each problem where all algorithms are specified in terms of increasing the value of an objective function subject to a constraint and where the algorithm class includes efficient algorithms as well as algorithms that behave correctly even in adverse circumstances; (*iii*) proposing a compositional model of the environment and agents which is more general than many of the models considered earlier; and (*iv*) providing a general approach with necessary conditions and sufficient conditions for application of the methodology.

The distributed systems model presented here deviates from most models used in the literature in two significant ways. First, common models in the literature, such as process networks, are static. By contrast, this paper presents algorithms that adapt to dynamic environments by exploiting whatever resources are available to them; the algorithms permit efficient computations in benign environments and computations that progress slower in adversarial environments. Second, most algorithms in the literature are specified in terms of a program for each agent. By contrast, algorithms in this paper are specified as goals for agents: each agent and group of agents has a utility (objective) function that they attempt to maximize subject to a constraint—the conservation law. Our design task is to identify an agent's utility function.

Many papers in the literature present efficient programs for agents that operate in specific types of environments. For most practical problems, such as designing distributed file systems, traditional approaches are preferable to designing systems of agents that attempt to improve their utilities by exploiting available resources. Systems specified in terms of goal-driven agents operating in dynamic environments, about which very little is assumed, help in developing theories of robust distributed systems.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: Numerical methods*. Prentice Hall, 1989.

[2] K. M. Chandy and M. Charpentier. Self-similar algorithms for dynamic distributed systems. Technical Report CS–TR–2007–001, California Institute of Technology, Mar. 2007.

[3] B. Charron-Bost. Agreement problems in fault-tolerant distributed computing. In *28th Annual Conference on Current Trends in Theory and Practice of Informatics*, volume 2234 of *LNCS*, pages 10–32, 2001.

[4] S. Chatterjee and E. Seneta. Towards consensus: some convergence theorems on repeated averaging. *Journal of Applied Probability*, 14(1):89–97, 1977.

[5] N. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, 1997.

[6] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[7] S. M. Pike and P. A. G. Sivilotti. Dining philosophers with crash locality 1. In *International Conference on Distributed Computing Systems (ICDCS'04)*, pages 22–29, 2004.

[8] W. Ren, R. W. Beard, and E. M. Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005 American Control Conference*, pages 1859–1864, 2005.

[9] K. Soharabi, J. Gao, V. Ailawadho, and G. Pottie. Protocols for self organization of a wireless sensor network. *IEEE Personal Communication Magazine*, 7(5):16–27, 2000.

[10] D. P. Spanos, R. Olfati-Saber, and R. M. Murray. Dynamic consensus on mobile networks. In *16th IFAC world congress*, 2005.

[11] N. Sridhar. Decentralized local failure detection in dynamic distributed systems. In *25th International Symposium on Reliable Distributed Systems (SRDS'05)*, pages 143–152, 2005.

[12] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stocastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, Sept. 1986.

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE