

# From Static Distributed Systems To Dynamic Systems

Achour Mostefaoui,  
IRISA, Université de Rennes 1, France

This work is based on results that appeared in the proceedings of IEEE SRDS'06

Co-authors : D. Agrawal, A. El Abbadi, S. Patterson, M. Raynal et C. Travers

# Outline

- Motivation
- Model of Static Systems
- Model of Dynamic Systems
- Consistency of Distributed Data
- Example Protocol: Leader Election
- How about the applications?
- Conclusion

# Motivation - P2P Systems

- P2P Systems originated as file-sharing applications
  - Best-effort semantics
- P2P Systems as full-fledged distributed computing platform?
  - Possible applications: grid computing
  - Need formal model and provably correct protocols

# P2P Systems - Challenges

- **Dynamicity**
  - Peers can join and leave the system at any time
  - Peers can fail
  - Stable clusters have been observed
- **Scalability**
  - Systems can grow to millions of peers
  - Impossible to maintain any global information

# Goals

- Develop a formal model for dynamic systems
- Provide methodology for translation of protocols designed for static distributed systems to dynamic systems
- Validate methodology through an example protocol translation

*This talk presents a first step towards defining a formal model for dynamic systems*

# Static Model

- Consists of  $n$  processes, some of which are faulty
  - Crash failures
- Reliable communication channels between all processes
  - Message eventually delivered if destination process does not fail
- System is asynchronous
  - No bound on time delay for communication or computation

# Protocols for Static Systems

- Assume every process has *a priori* knowledge of  $n$
- Progress Condition: Progress is guaranteed as long as **no more than  $f$**  processes are faulty

$n$  and  $f$  are key parameters in protocols for static distributed systems

# Communication Primitives

- Query-Response
  - Broadcast a query to all processes
  - Wait for *winning* responses from  $n - f$  processes
  - If no more than  $f$  process are faulty, query is not blocked
- Reliable Broadcast
  - Message is sent to to all processes
  - If message is delivered by a process, then it is delivered by all correct processes

# Dynamic Systems

- Same asynchrony assumptions as for static model
- **Finite arrival model** : system has infinitely many processes, but each run has only finitely many

(Aguilera '04, Merritt and Taubenfeld '00)

- Processes may join and leave at any time
  - Fail-stop model for node departures

*n* and *f* do not exist

# Dynamic Process Model

- Replace notion of correct processes with  
***STABLE*** : the set of processes that once having entered the system, neither crash nor leave
- Replace ***n*** and ***f*** by  **$\alpha$**  : a lower bound on the size of ***STABLE***
- New progress condition:  
Progress is guaranteed as long as  
 **$|STABLE| \geq \alpha$**

# Dynamic Model - Query/Response

- Query initiator broadcasts message to all processes
- Any process that receives query sends response
- Query initiator waits for  $\alpha$  winning responses
- If  $|\mathbf{STABLE}| \geq \alpha$ , query is not blocked indefinitely

# Dynamic Model - Reliable Broadcast

- Cannot expect broadcast to reach all processes
  - Process may crash before receiving message
  - Process may join after broadcast has terminated
- Can we expect broadcast to reach all processes that do not crash?
  - Need special broadcast primitive for dynamic systems

# Persistent Reliable Broadcast

(Friedman et al. 05)

- Send message to *sufficiently large subset* of processes so that message is not lost by system if some processes crash
- Guarantees that every process in **STABLE** will receive
  - Message that was broadcast OR
  - Message of the same type that was broadcast later
- What about new processes?
  - Broadcasts inquiry
  - Every process responds with state and logical date of last message delivered
  - New process adopts state with most recent logical date

# Summary of Dynamic Model

Static

---

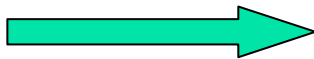
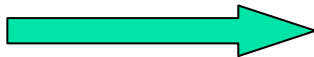
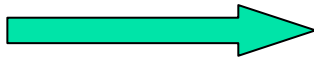
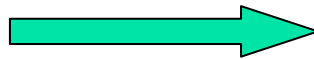
$p$  is correct

$n - f$

QR:

$n - f$  winning responses

Reliable  
Broadcast



Dynamic

---

$p \in \text{STABLE}$

$\alpha$

QR:

$\alpha$  winning responses

Persistent Reliable  
Broadcast

# Consistency of Distributed Data

- Need of replication
  - Efficiency reasons
  - Fault-tolerance
- Replication needs access services
  - Read/Write operations
  - Atomic operations (e.g. atomic commit)

# Static Model (1)

- Read/Write operations
  - Quorums (majority, ROWA, votes, etc.)
  - Assumption:  $f < n / 2$
- Atomicity
  - Atomic broadcast, NBAC, leader election, etc.
  - FLP result (1985): **Undecidable**

# Static Model (2)

Atomicity is undecidable

- Weakening the problem
  - Probabilistic agreement, k-set agreement
- Strengthening the system
  - Partial synchrony, Failure detectors
- Imposing conditions on the entries
  - Condition-based agreement (Mostefaoui et al. – Journal ACM'03)

# Dynamic Model

- Read/Write operations
  - Quorums: ?
  - Assumption:  $f < n / 2$
- Atomicity
  - Failure detectors: **scope ?**
  - Partially synchronuous: **realistic ?**

# Atomicity - example: Leader election

Every process has a `leader()` function that returns a process name

**Eventual Leadership:** There is a time  $t$  and a correct process  $p$  such that, after  $t$ , every invocation of *leader()* by any correct process returns  $p$

# Leader Election in a Static System

- There is no protocol to elect a leader in a fully asynchronous static system
  - Due to impossibility of consensus
- Need to add additional assumptions to solve problem
  - Eventual synchrony
  - Message exchange pattern
- Use a protocol based on second approach  
(Mostefaoui-Raynal-Travers SRDS '04)

# Additional Assumption for Static System

$MP_{static}$ : There is a time  $t$ , a correct process  $p$ , and a set  $Q$  such that for all  $t' \geq t$

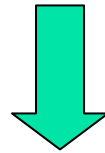
1.  $|Q| = f + 1$
2. Every process in  $Q$  receives a winning response from  $p$

# Electing a Leader in a Static System

- Each process keeps track of list of trusted processes
- Processes gossip to share trusted lists
- Eventually,
  - Every process in  $Q$  always receives a winning response from  $p$
  - Every process always receives a winning response from some member of  $Q$  (since  $|Q| = f + 1$ )
- Eventually, all processes has intersecting trusted lists
- Leader  $p$  elected from trusted lists

# Leader Election in a Dynamic System

**Eventual Leadership:** There is a time  $t$  and a **correct process**  $p$  such that, after  $t$ , every invocation of  $leader()$  by any **correct** process returns  $p$



**Eventual Leadership:** There is a time  $t$  and a **process**  $p \in STABLE$  such that, after  $t$ , every invocation of  $leader()$  by any process returns  $p$

## Translating Assumption for Dynamic System

$MP_{dyn}$  : There is a time  $t$ , a process  $p \in STABLE$ , and a set  $Q$  such that for all  $t' \geq t$

1.  $Q \subseteq up(t')$
2. Every process in  $Q$  receives a winning response from  $p$
3. Every process receives a winning response from some member of  $Q$

# Electing a Leader in a Dynamic System

(Mostefaoui et al. SRDS'05)

- Each process keeps track of list of trusted processes
- Processes gossip to share trusted lists
- Eventually,
  - Every process in  $Q$  always receives a winning response from  $p$
  - Every process always receives a winning response from some member of  $Q$
- Eventually, every process has the same trusted list
- Leader  $p$  elected from trusted list

# How about the application level?

- Chemical abstraction paradigm
  - Gamma programming model :  
*Multiset transformation language*
  - Tuple Space (Linda in context)
  - Java Space

*Max*:  $x, y \rightarrow y \Leftarrow (x \leq y)$

*Sort*:  $(i, v), (j, w) \rightarrow (i, w), (j, v) \Leftarrow (i < j) \wedge (v > w)$

# Multiset Transformation

- Replace **read/write** by **Insert/Remove/Lookup**
- Replace **variables** by **values**
- Lookup needs accessibility
  - Indexation
- Insert needs persistency
  - Replication
- Atomicity:
  - Remove one value: agreement services
  - **Remove a group of values: agreement services**

# Is perfect agreement mandatory?

- Remove:
  - k - set agreement
  - $\epsilon$  - agreement
  - Condition-based agreement
  - Probabilistic agreement
- Lookup:
  - Renaming (Attiya et al. Journal ACM'90)  
Less difficult to solve than consensus  
and eases the implementation of atomicity

# Conclusion

- Models for classical static distributed systems not applicable to P2P systems
- There is need for new dynamic model of distributed systems
- Is there a methodology for translating protocols for static systems to dynamic systems?
- The presented material represents a first step towards the definition of a formal model for dynamic systems