Approach

- Results of different designs (via computational methods)
- Why?
- Approach

1. For 50,000+ clients, it delivers high aggregate performance.
- Indexed transactions
- Fault-tolerance
- Scalability

2. It provides distributed archival
- GIS: a scalable distributed archival for large
- Foundations on Distributed Systems
- GFS, Hadoop, etc.

Project:
- CE 539 (Fall 05)
1. Done procedure.
2. Only if each frame is.
R.L.R.
and end to both
L make copy of M

Natural Reliable Broadcast Protocol

1. Send M to F
2. Discard F
3. Send M to F
4. Send M to F
5. Send M to F

Get at the answer time
complete certainty that they will both 먼저
lose to bounded protocol can summarize
ik.

Two General Solutions: if necessary can be

Distributed Algorithms

message if helps/progress
Let's analyze the diagram and textual content:

- **Performance**: Performance is about minimizing the number of steps.
- **Prefix**: Prefix means that a receiver receives a prefix message.
- **Recall**: Recall means that a receiver receives a recall message.
- **Link**: Links are represented by arrows and indicate the direction of communication.

The diagram shows a feedback loop with two main steps:

1. **Performance correction**:
   - **Send** the link message.
   - **Receive** the link message.

2. **Performance**:
   - If the link message is received, perform the correction.
   - If no link message is received, proceed with the normal function.

The feedback loop continues until the performance is corrected or the system reaches a stable state.
\[ P(N) : \text{prob. that all rec. receive.} \]

\[ P^n \]

\[ P + (1-P)P + (1-P)P^2 \]

\[ \text{prob that at least one receiver doesn't receive} \]

\[ 1 - P^n \]

\[ \text{how many attempts are needed} \sim \frac{1}{P^n} \]

\[ \text{Avg \# 1 attempt} \]

**Time in Distributed System**

- **NTP**
  - granularity of syn. t.e
  - drift

- Using time stamps based on physical clocks is problematic.

- Problem in determining causality

\[ t_1 \text{ start} \rightarrow \text{rec. crash} \rightarrow t_2 < t_1 \]
Lamport's Clock - causality

- happened before →
  1. On some processor i)
     e2 follows e1, then e1 → e2
  2. if e1 is send event for
     a message M and e2 is
     its corresponding receiv event
     then e1 → e2
  3. Transitive closure
     e1 → e2, e2 → e3
     then e1 → e3

Implementation
- Each processor Pi maintain
  counter Ci (initialised to 0)

IRi: between any two events on the
same processor say Pi, Ci
IRi:

IRi:
Check time stamp is 
+ break. If c < 
process 1, it is used

If (s, p) \( \in \Pi \) \( \text{and} \) \( \Pi (e) > \Pi (e') \), then 

If (e') \( \in \Pi \) \( \text{and} \) \( \Pi (e) > \Pi (e') \), then 

This will only give a partial ordering unless

\( T(\text{end}) = c_i = \max (c_i, T(\text{end})) + 1 \) :

Upon receipt of H on proc. P i 

\( \text{End} \) 

With each message received, M 

\( T (\text{end}) \) 

9/6/17
Lamport's Clock can be used for distributed mutual exclusion.

Ref: Time, Clocks and the ordering of events in a Distributed System, Leslie Lamport
CACM Jul. '78