Object replication in distributed systems

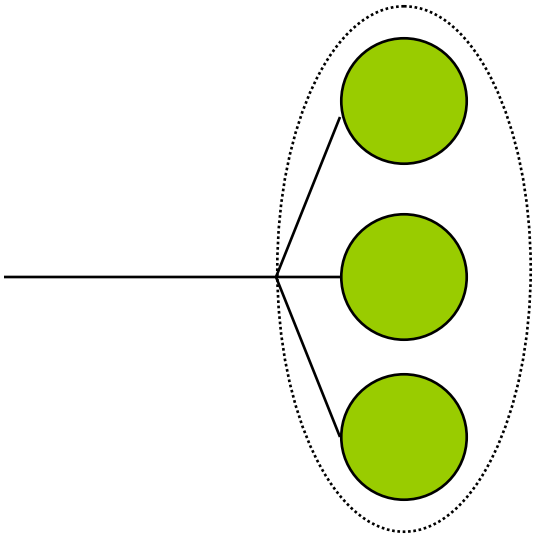
Mark C. Little,
Department of Computing Science,
University of Newcastle

Overview

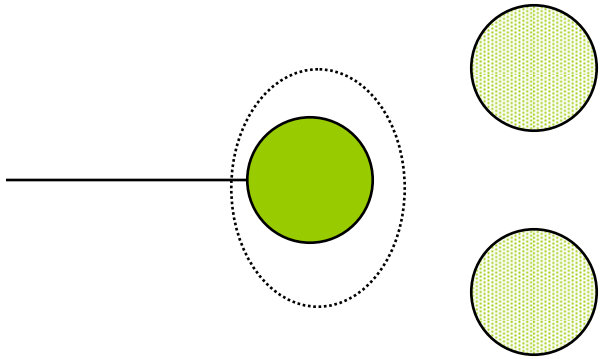
- descriptions of replication protocols
  - where these protocols can and cannot be used
- the Arjuna approach to supporting replication
- replica configuration issues
- the Replica Management System
- replicating composite objects
- implementation experience
- conclusions

Replication protocols

– there are essentially two categories of replication protocol:
  - active replication
  - single-copy passive replication
– strong and weak consistency



Active



Single-copy passive

Protocol descriptions

- active replication:
  - assume objects are deterministic
  - requires group communication mechanism to deliver the same set of messages to each active replica in the same order
  - often the preferred choice where masking of replica failures with minimum time penalty is highly desirable
  - can be used to mask K:
    - permanent omission, value and omission, and timing failures with K+1 replicas
    - arbitrary (Byzantine) failures with 2K+1 or 3K+1 replicas
  - 2K+1 replicas to tolerate network partitions

Protocol descriptions

- passive replication:
  - need not require complex, order preserving group communications protocols
    - can be implemented using traditional RPC
    - easily ported to other environments
  - performance in the presence of primary failures can be substantially poorer than under no failures
  - can be used to mask K:
    - permanent omission failures with K+1 replicas
  - tolerate network partitions with 2K+1 replicas

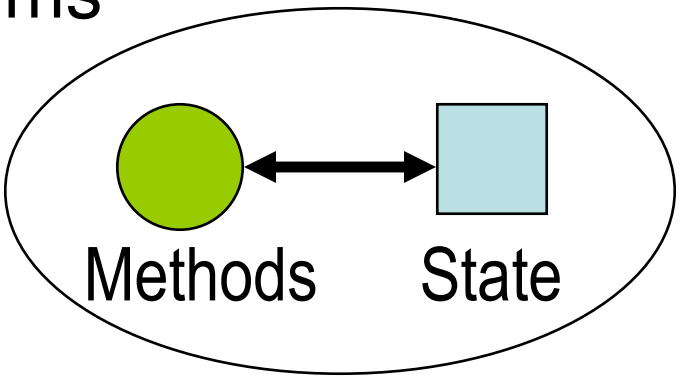
Providing replication

- although all objects could be replicated passively, performance and failure masking problems rule this out
- in general we require a suite of replication protocols:
  - primary copy replication
  - available copies
  - weighted voting
  - coordinator-cohort
- select protocol on a per-object (class) basis

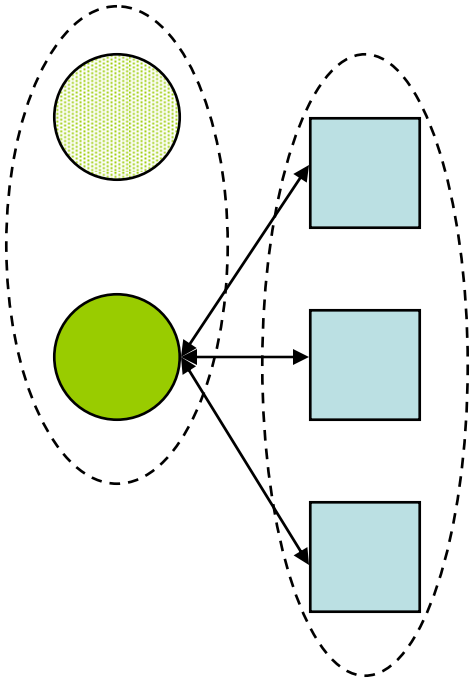
Replication infrastructure

– it is possible to provide an infrastructure which supports all replication protocols

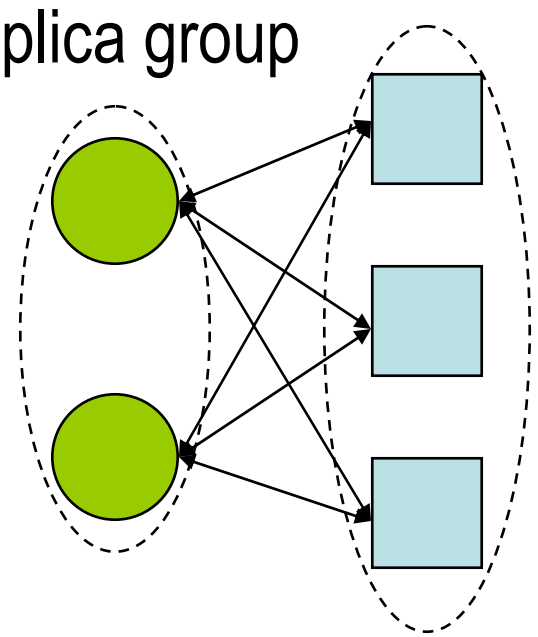
– separate object into methods and state, and provide appropriate naming and binding mechanisms



Replication protocols



Passive Replication



Active Replication

Arjuna's replication protocol

- default protocol is based upon single-copy passive replication
  - multiple instances of an object's methods, but only the primary is active
    - primary failure requires action to abort and restart
  - multiple instances of persistent state are updated
    - action can commit as long as a single instance remains available
- naming service is implemented using active replication
  - provides toleration of network partitions for all replica groups

Implementation

```
class StateManager
{
public:
    virtual Boolean save_state (ObjectState&) = 0;
    virtual Boolean restore_state (ObjectState&) = 0;

    virtual Boolean hasRemoteState () const;

    Boolean setStoreInformation ("replica configuration");
    const Uid& get_uid () const;

    const TypeName type () const;
};
```

Active replication protocol

- experimented with active replication protocol
  - K-resilient
- uses reliable group communications protocol
  - replicated RPC
- atomic actions used to impose ordering only where required
  - concurrency control allows multiple readers to be interleaved
- have client and server groups
- flow control and timeouts

Replica configuration issues

- how to arrive at the optimum number and location of replicas?
- availability is not necessarily proportional to the number of replicas
- replica configuration depends upon:
  - failure characteristics of the distributed environment
    - component inter-dependencies
  - read/write ratio of interactions with the object
  - desired quality of service
    - trade-off between availability and performance
  - object inter-dependencies

Replica management system

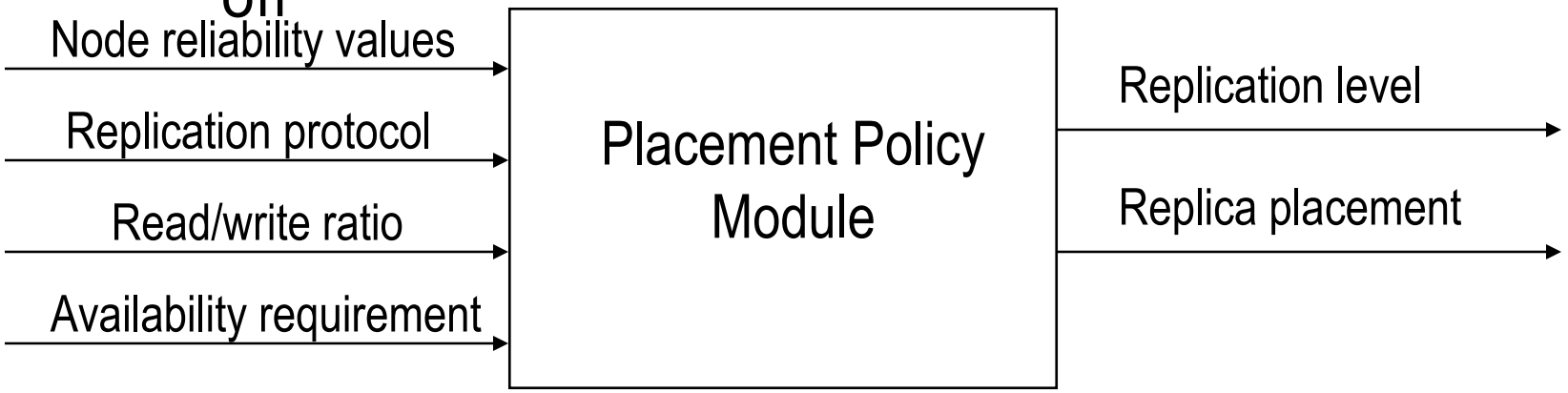
- measure *attribute values*:
  - component reliability
    - nodes
    - communication links
  - inter-dependencies
    - components
    - objects
  - performance values
    - nodes
    - communication links

Replica management system

- system administrator
- monitor daemon
  - MTTF and MTTR
    - cause of “failure”
  - performance
    - architecture, configuration, etc.
- dependency tracker
  - collate information from all monitor daemons
  - determine *node availability*
  - use common failures to infer dependencies
    - more complex algorithms may yield better results
- object management module

Placement policy

- computes number and placement of replicas
  - supply user's desired QoS
  - rank availability and performance in case of trade-off



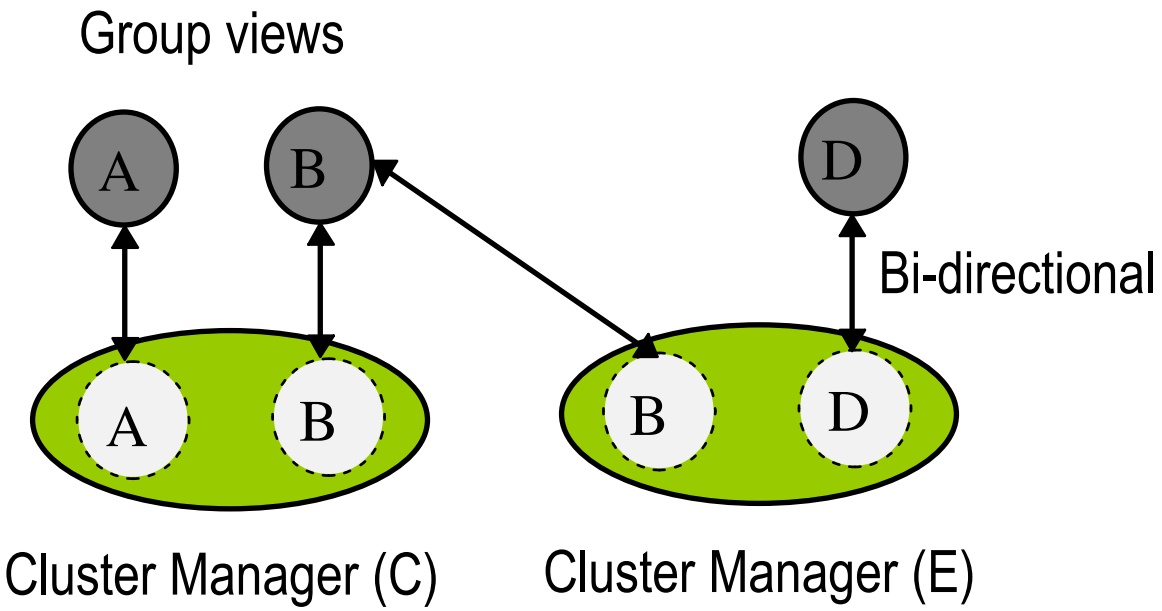
Replicating composite objects

- create two new types of group view at the naming service:
  - *clustered replica groups*
    - collections of group views are clustered together
    - obtaining any one member of the cluster caches all cluster information at the client
  - *template replica groups*
    - common information is factored into a template
    - only store unique information on a per group basis
    - *wildcard template*

Clustered replica groups

- related replica groups can be clustered within the naming service
  - Each cluster is managed by a separate Cluster Manager object
    - appears as another group view within the naming service
    - user's cannot determine whether a group identifier belongs to a cluster or a single group
- replica groups can be accessed:
  - directly, through their group identifier
  - indirectly, through the Cluster Manager's identifier
- all cluster members are cached within the client for the duration of the atomic action

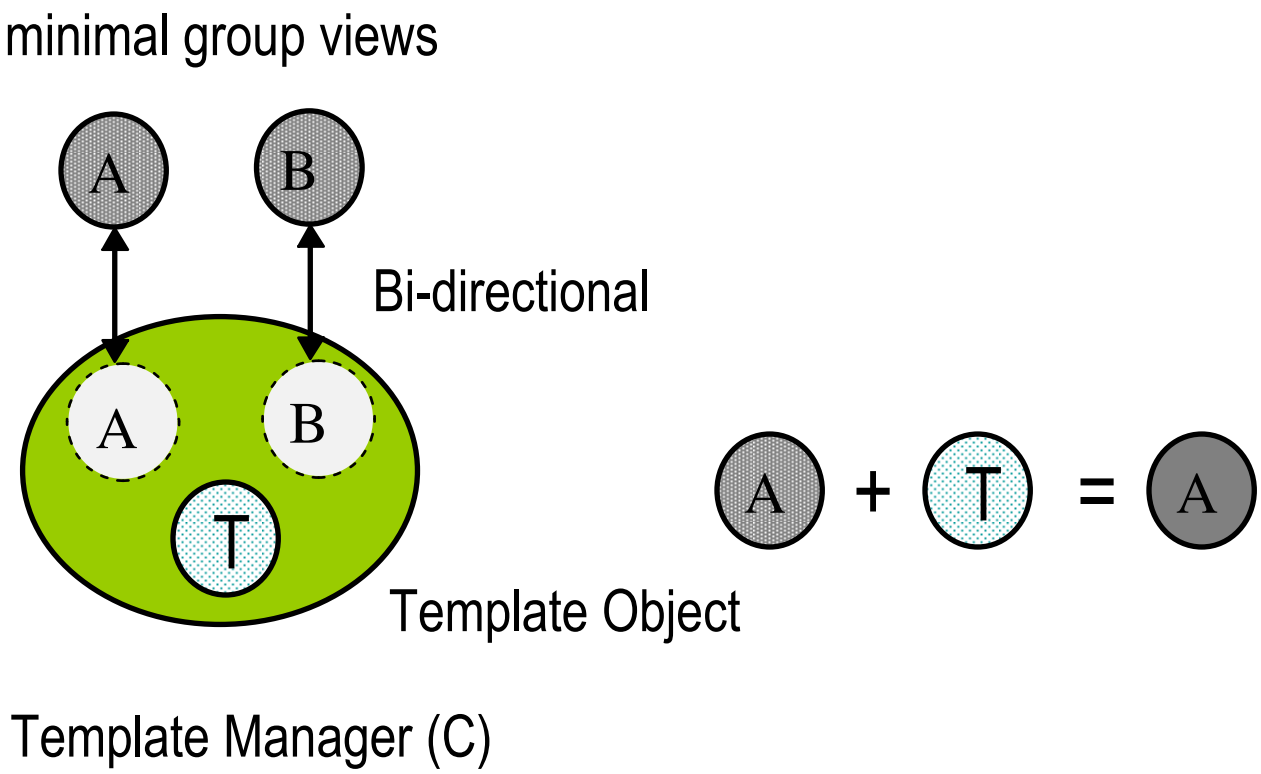
Clustered replica groups



Template replica groups

- related objects are typically identically replicated, e.g., composite objects
  - efficiency of the naming service can be improved by reducing the amount of information it must store
- factor common information into a template, and associate group specific information with it
  - apply template to obtain group view at the client, rather than the naming service
    - reduces amount of network traffic
- wildcard template can be used to replicate every object

Template replica groups



Implementation experience

- University's student registration system
  - has been used successfully since 1994
  - 100+ simultaneous users
  - 12,000+ students registered in 5 days
  - each student record is a separate Arjuna object
    - persistent states are replicated 3 times
    - methods are replicated 5 times
  - performance is well within the University's requirements, even at peak load
  - several machine crashes have occurred, and the system masked them without most users noticing

Future directions

- large-scale
  - weak consistency
- merge cacheing with replication
- further modularisation
  - state
  - concurrency
  - consistency
  - input/output
- virtual synchrony

Conclusions

- systems require more than a replication protocol
  - suite of replication protocols
  - RMS
- object dependencies can improve availability and performance
- inheritence aids usability
- appropriate choice of default protocol
  - performance is disk bound in the absence of failures
- useful in conjunction with atomic actions