IPC In Distributed Operating Systems

CS 239, Spring 1995

May 16, 1995

Outline

- Distributed implications for RPC
- Distributed shared memory
- Causally ordered multicast systems

RPC In Distributed Operating Systems

- Finding remote servers
- Data transport mechanisms
- Handling heterogeneity

Finding Remote RPC Servers

In the general case, a caller's local site will not know the location of the called program

How to find it?

- Broadcast
- Intelligent multicast
- Consult a server

Finding Remote Servers in Unix RPC

Consult the Network Information Service (NIS)

A naming service that keeps track of a variety of different name-to-location mappings

Including names for RPC servers

Characteristics of the NIS

- Server daemons to consult server's database
- Client site daemons to forward local requests to appropriate server daemon
- Databases are distributed
- And replicated with weak consistency

Getting Server Information From NIS

Several system calls consult the local client daemon

When possible, it answers them locally

When not, it makes a remote request

Setting Up an RPC Connection Via the NIS

- Server runs on a particular port on a particular machine
- Client uses NIS to find out where the machine is (given, say, its name)
  - Need to find IP address
- The port is already unique within that machine

Finding Remote Servers in Mesa RPC

Uses Grapevine system

Grapevine is an optimistically replicated name server

Servers register locations with Grapevine

Clients look up services by name

Data Transport Mechanisms for RPC

Really transparent to RPC facility

Just getting bits from here to there

But choice can strongly affect certain RPC characteristics

Like performance

And security

RPC Transport Characteristics

- Generally small amounts of information transported
  - But not always
- Communications tend to be self-contained
- Sender tends to block waiting reply
  - But another process may run at sending end
- You may care about security

Security for RPC

RPC data transport is subject to the same security problems other data transports are

Eavesdropping, corruption, masquerading, replay, etc.

For complete safety, encryption and authentication must be applied

Encryption and Authentication for RPC

How many keys?

- One for all RPC communications?
- One per machine?
- One per RPC server?
- One per client/server pair?

And how to distribute them?

Unix RPC "Security"

- No encryption built in
- Several levels of authentication provided
  - None
  - UID/GID attached (but not verified)
  - DES-based authentication (not very strong)

Heterogeneity and RPC Mechanisms

RPC is meant to be largely transparent

- Don't worry about location of server
- And don't worry about other server characteristics

What if the server is a different type of machine?

Unix RPC and XDR

XDR - External Data Representation

A library that deals with heterogeneity

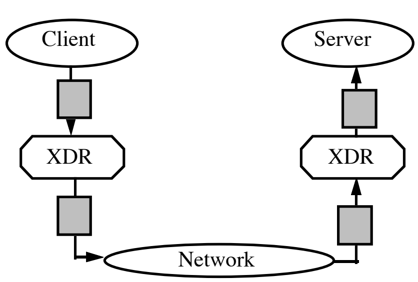
Based on an ARPA standard

- Similar to an ISO standard

Basic idea - data travels between client and server in a canonical format

- Each machine translates to local format

XDR Diagram



Who Uses XDR?

Must be built into the application - Unix RPC does not use it by default

But protocol compilers typically do the hard work of using XDR for real programs

Must be sure to use it consistently

Another Option for Handling Heterogeneity

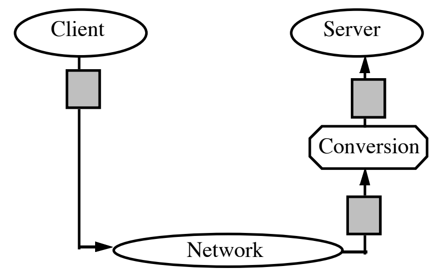
Most modern machines don't use heterogeneous data formats

So XDR translations may be wasted work most of the time

Sometimes *receiver-makes-right* is used instead

Puts burden on non-conformists

Receiver-Makes-Right Diagram



Synchronization for Cooperating Processes

If a group of cooperating processes need to see a common view of the world, what can we do?

Various approaches, including

- Distributed shared memory
- Causally ordered message multicasting

Distributed Shared Memory

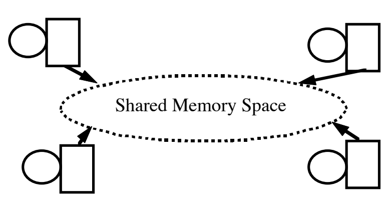
Cooperating processes share a coherent memory space

Can this be provided between processes that cannot access the same physical memory?

Yes - and sometimes at a reasonable cost

Distributed shared memory must be highly transparent to be successful

Distributed Shared Memory Diagram



Approaches to DSM

- Central server
- Migration
- Read replication
- Full replication

Central Server DSM

A central server maintains all data pages

Returns data on reads

Makes updates on writes

Care needed to avoid duplicating writes

Server duties can be split between several sites

Migration DSM

Data lives where most recently used

When off-site memory is requested, it migrates to the requesting site

Typically at block or page levels

With locality of reference, works well

But possibility of thrashing

Servers keep track of data locations

Read-Replication DSM

Multiple copies of readable blocks stored

If block is written, invalidate other copies

Each block has owner that permanently stores it

If writes frequent, invalidation can be complicated and expensive

Full-Replication DSM

Full-service replicas of data blocks/pages stored at several machines

Requires consistency mechanism

- For same reasons as replicated files
- But different mechanisms may be appropriate

Memory Coherence in Replicated DSM Schemes

If more than one copy of a data item exists, problems arise when data is written

Consistency model describes what level of consistency a DSM system provides

Consistency Models for DSM

- Strict consistency(like if only one copy)
- Sequential consistency (like serializability)
- General consistency (eventual consistency)
- Processor consistency (per machine)
- Weak consistency (up to programmer)

Protocols to Provide DSM Coherence

- Write-invalidate protocol
  - Invalidate other copies on write
  - Expensive for those other copies
- Write-update protocol
  - Update other copies on write
  - Higher network traffic results

Controlling Thrashing in DSM

How to prevent 2 sites sharing a page for write from thrashing the net?

One way is to lock a written page down at a site for some period Δ

If another site wants to write it, that site must wait for Δ to expire

Used in Mirage system

Granularity in DSM

How big should the unit of shared memory be?

Should be related to hardware's page size

Pros/Cons of Large Pages for DSM

+ Takes advantage of locality of reference
+ Minimizes cost of running paging code
+ Can minimize communications cost

– Leads to greater contention

Pros/Cons of Small Page Size for DSM

+ Less contention
  Minimizes *false sharing*
+ Lower cost to move small data objects

– Greater overheads

Page Replacement in DSM

When a local node needs to bump a shared memory page, what does it do?

LRU alone not a sufficient answer

Must consider private vs. shared pages

  And replication issues

  And read-only pages

LRU within classes often used

Some DSM Systems

- Ivy (Yale)
- Mirage (UCLA)
- Clouds (Georgia Tech)
- Munin (Rice U.)
- Memnet (U. of Delaware)

Another Way to Provide Distributed Consistency

Assume a set of cooperating processes start out consistent

With a distributed state

And assume the only interactions are through messages

If all processes see the messages in the same order, all see a consistent state

Causal Ordering

Messages carry updates, allow reading data

All messages must be handled in the appropriate order to guarantee consistency

- Absolute ordering
- Causal ordering

Easy to provide synchronously

Hard to provide asynchronously

Isis - A Causal Message Ordering Service

Isis is a library for guaranteeing message ordering

By multicast operations

Can provide either ABCAST or CBCAST

Using low-overhead mechanisms

The Isis Model

- Set of processes $P=\{p_1,p_2, \ldots, p_n\}$
- Process failure is detectable
- Processes structured in process groups $G=\{g_1,g_2, \ldots\}$
- Process groups are named
- Processes can belong to multiple groups
- Multicasts only go to members in sender's group

Views in Isis

- The membership of a given group at a given moment is a *view*
- A *view sequence* of a process group is a list of its views
  - Each view differs from last by one process
- Processes learn of failures only by requesting view of a group

Isis Message Transport

Isis is built on top of a transport mechanism

With the following characteristics

- Lossless
- Uncorrupted
- Sequenced
- Transport layer intercepts and discards messages from failed processes

Isis Message Primitives

- *$send_p(m)$*

  Denotes sending of *m* by *p*; atomic
- *dests(m)*

  Set of processes receiving message *m*
- *$rcv_p(m)$*

  Event of process *p* receiving message *m*

  "Receive" is not the same as "deliver"

Events in Isis

- The execution of a process is a partially ordered sequence of events
- Events consist of things like sending a message, receiving a message
- Acyclic event order reflects the dependence of events at p

$\xrightarrow{p}$

Virtual Synchrony

Users program Isis as if it scheduled one distributed event at a time

How to achieve this?

1). Schedule one distributed event at a time

2). Enforce degree of synchrony actually needed for each case to give this appearance

Aspects of Virtual Synchrony

- Address expansion

  Ensure a given group multicast goes to all members of a given view
- Delivery atomicity and order

  Fault tolerant delivery

  Multiple destinations receiving same messages get them in consistent order

Vector Clocks

Similar to version vectors, kept per process

For P processes, version clocks are of length P

$VT(p_i)$ is the version clock of p_i

$VT(p_i)[i]$ is i^{th} component

Assign a unique, increasing timestamp to each send event

Properly handled, version clocks encode causality

The CBCAST Protocol

Before multicasting message m, process p_i increments $VT(p_i)[i]$

When p_j receives message m from p_i, delay delivery until

For all k, $VT(m)[k] = VT(p_j)[k]+1$ if $k=i$

$VT(m)[k] <= VT(p_j)[k]$ otherwise

On delivery of m, update j's vector time

What's Going on Here?

Don't deliver a message to a process until:

1). The process has received all earlier messages from the sending process

2). The process has received all earlier messages from third party processes received by the sending process

Can CBCAST Deadlock?

Deadlocks are based on waiting

- So when does CBCAST wait?

  When a message has arrived at a process, but cannot yet be delivered

- What is it waiting for?

  Other messages to arrive

Is CBCAST's Message Waiting Safe?

The messages CBCAST waits for are messages that were delivered to (or originated with) the sender before the message m that's waiting

So they are not waiting for message m

Reliable delivery guarantees that the messages being waited for will arrive

Other Issues in Isis

- Implementing ABCAST
- Delivery atomicity
- Handling group membership changes
- Minimizing communications for overhead
- Interactions of multiple overlapping groups
- Performance, performance, performance