Remote Procedure Call

Outline
Protocol Stack
Presentation Formatting

RPC Motivation

- Program distributed systems just like undistributed ones
  - \( r = p(x,y,z) \) locally becomes
  - \( r = p(dest, x, y, z) \) remotely
- Problems
  - Different failure modes
  - Different performance characteristics
    - Elapsed time
    - (no) shared memory

Network support of RPC

- Not looking at language run-time support of RPC
  - Type checking
  - Transparent remote invocation at the language level
- Instead
  - The between-machines issues:
    - Fragmentation
    - Matching replies with requests/Reliable delivery
    - Indicating to the remote machine what service is required of it

RPC Components

- Protocol Stack
  - BLAST: fragments and reassembles large messages
  - CHAN: synchronizes request and reply messages
  - SELECT: dispatches request to the correct process
- Stubs
Bulk Transfer (BLAST)

- Unlike AAL and IP, tries to recover from lost fragments

- Strategy
  - selective retransmission
  - aka partial acknowledgements, aka selective acknowledgements

BLAST Details

- Sender:
  - after sending all fragments, set timer DONE
  - if receive SRR, send missing fragments and reset DONE
  - if timer DONE expires, free fragments

BLAST Details (cont)

- Receiver:
  - when first fragments arrives, set timer LAST_FRAG
  - when all fragments present, reassemble and pass up
  - four exceptional conditions:
    - if last fragment arrives but message not complete
      - send SRR and set timer RETRY
    - if timer LAST_FRAG expires
      - send SRR and set timer RETRY
    - if timer RETRY expires for first or second time
      - send SRR and set timer RETRY
    - if timer RETRY expires a third time
      - give up and free partial message

BLAST Header Format

- MID must protect against wrap around
- TYPE = DATA or SRR
- NumFrags indicates number of fragments
- FragMask distinguishes among fragments
  - if Type=DATA, identifies this fragment
  - if Type=SRR, identifies missing fragments
Request/Reply (CHAN)

- Guarantees message delivery
- Synchronizes client with server
- Supports *at-most-once* semantics

Simple case

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Request 1</td>
</tr>
<tr>
<td>ACK</td>
<td>ACK 1</td>
</tr>
<tr>
<td>Reply</td>
<td>Request 2</td>
</tr>
<tr>
<td>ACK</td>
<td>Reply 2</td>
</tr>
</tbody>
</table>

Implicit Acks

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request 1</td>
<td>Request 2</td>
</tr>
<tr>
<td>ACK 1</td>
<td>ACK 2</td>
</tr>
<tr>
<td>Reply 1</td>
<td>Reply 2</td>
</tr>
</tbody>
</table>

CHAN Details

- Lost message (request, reply, or ACK)
  - set RETRANSMIT timer
  - use message id (MID) field to distinguish
- Slow (long running) server
  - client periodically sends “are you alive” probe, or
  - server periodically sends “I’m alive” notice
- Want to support multiple outstanding calls
  - use channel id (CID) field to distinguish
- Machines crash and reboot
  - use boot id (BID) field to distinguish

CHAN Header Format

```c
typedef struct {
    u_short Type; /* REQ, REP, ACK, PROBE */
    u_short CID; /* unique channel id */
    int MID; /* unique message id */
    int BID; /* unique boot id */
    int Length; /* length of message */
    int ProtNum; /* high-level protocol */
} ChanHdr;
```

Synchronous vs Asynchronous Protocols

- Asynchronous interface
  ```c```
  xPush(Sessn s, Msg *msg)
  xPop(Sessn s, Msg *msg, void *hdr)
  xDemux(Prot1 hlp, Sessn s, Msg *msg)
  ```c```
- Synchronous interface
  ```c```
  xCall(Sessn s, Msg *req, Msg *rep)
  xCallPop(Sessn s, Msg *req, Msg *rep, void *hdr)
  xCallDemux(Prot1 hlp, Sessn s, Msg *req, Msg *rep)
  ```c```
- CHAN is a hybrid protocol
  - synchronous from above: `xCall`
  - asynchronous from below: `xPop`/`xDemux`
chanCall(Sessn self, Msg *msg, Msg *rmsg)
{
    ChanState *state = (ChanState *)self->state;
    ChanHdr *hdr;
    char *buf;

    /* ensure only one transaction per channel */
    if ((state->status != IDLE))
        return XK_FAILURE;
    state->status = BUSY;

    /* save copy of req msg and ptr to rep msg*/
    msgConstructCopy(&state->request, msg);
    state->reply = rmsg;

    /* fill out header fields */
    hdr = state->hdr_template;
    hdr->Length = msgLen(msg);
    if (state->mid == MAX_MID)
        state->mid = 0;
    hdr->MID = ++state->mid;

    /* attach header to msg and send it */
    buf = msgPush(msg, HDR_LEN);
    chan_hdr_store(hdr, buf, HDR_LEN);
    xPush(xGetDown(self, 0), msg);

    /* schedule first timeout event */
    state->retries = 1;
    state->event = evSchedule(retransmit, self, state->timeout);

    /* wait for the reply msg */
    semWait(&state->reply_sem);

    /* clean up state and return */
    flush_msg(state->request);
    state->status = IDLE;
    return state->ret_val;
}

retransmit(Event ev, int *arg)
{
    Sessn s = (Sessn)arg;
    ChanState *state = (ChanState *)s->state;
    Msg tmp;

    /* see if event was cancelled */
    if ( evIsCancelled(ev) ) return;

    /* unblock client if we've retried 4 times */
    if (++state->retries > 4) {
        state->ret_val = XK_FAILURE;
        semSignal(state->rep_sem);
        return;
    }

    /* retransmit request message */
    msgConstructCopy(&tmp, &state->request);
    xPush(xGetDown(s, 0), &tmp);

    /* reschedule event with exponential backoff */
    evDetach(state->event);
    state->timeout = 2*state->timeout;
    state->event = evSchedule(retransmit, s, state->timeout);
}

chanPop(Sessn self, Sessn lls, Msg *msg, void *inHdr)
{
    /* see if this is a CLIENT or SERVER session */
    if (self->state->type == SERVER)
        return(chanServerPop(self, lls, msg, inHdr));
    else
        return(chanClientPop(self, lls, msg, inHdr));
}
```c
chanClientPop(Sessn self, Sessn ll, Msg *msg, void *inHdr)
{
    ChanState  *state = (ChanState *)self->state;
    ChanHdr    *hdr = (ChanHdr *)inHdr;
    /* verify correctness of msg header */
    if (!clnt_msg_ok(state, hdr))
        return XK_FAILURE;
    /* cancel retransmit timeout event */
    evCancel(state->event);
    /* if ACK, then schedule PROBE and exit*/
    if (hdr->Type == ACK)
    {
        state->event = evSchedule(probe, s, PROBE);
        return XK_SUCCESS;
    }
    /* save reply and signal client */
    msgAssign(state->reply, msg);
    state->ret_val = XK_SUCCESS;
    semSignal(&state->reply_sem);
    return XK_SUCCESS;
}
```

### Housekeeping 10/22

- Jacobsen/Karels with and without timestamps
- The Karn/Partridge and Jacobsen/Karels papers are available from the ACM Digital Library. Go to [www.wsulibs.wsu.edu](http://www.wsulibs.wsu.edu), choose the Articles Indexes/E-Journals tab, choose ACM Digital library from the list. Search for authors names: must either be on wsu network or use wsulibs proxy server.

### Simple RPC Stack

```
SELECT
  CHAN
    BLAST
      IP
        ETH
```
SunRPC

- IP implements BLAST-equivalent
  - except no selective retransmit
- SunRPC implements CHAN-equivalent
  - except not at-most-once
- UDP + SunRPC implement SELECT-equivalent
  - UDP dispatches to program (ports bound to programs)
  - SunRPC dispatches to procedure within program

SunRPC Header Format

- XID (transaction id) is similar to CHAN’s MID
- Server does not remember last XID it serviced
- Problem if client retransmits request while reply is in transit

Presentation Formatting

- Marshalling (encoding) application data into messages
- Unmarshalling (decoding) messages into application data
- Data types we consider
  - integers
  - floats
  - strings
  - arrays
  - structs
- Types of data we do not consider
  - images
  - video
  - multimedia documents

Difficulties

- Representation of base types
  - floating point: IEEE 754 versus non-standard
  - integer: big-endian versus little-endian (e.g., 34,677,374)
- Compiler layout of structures
Taxonomy

- Data types
  - base types (e.g., ints, floats); must convert
  - flat types (e.g., structures, arrays); must pack
  - complex types (e.g., pointers); must linearize

- Conversion Strategy
  - canonical intermediate form
  - receiver-makes-right (an $N \times N$ solution)

eXternal Data Representation (XDR)

- Defined by Sun for use with SunRPC
- C type system (without function pointers)
- Canonical intermediate form
- Untagged (except array length)
- Compiled stubs

```c
#define MAXNAME 256;
#define MAXLIST 100;

struct item {
    int    count;
    char   name[MAXNAME];
    int    list[MAXLIST];
};

bool_t xdr_item(XDR *xdrs, struct item *ptr)
{
    return(xdr_int(xdrs, &ptr->count) &&
        xdr_string(xdrs, &ptr->name, MAXNAME) &&
        xdr_array(xdrs, &ptr->list, &ptr->count, MAXLIST, sizeof(int), xdr_int));
}
```

<table>
<thead>
<tr>
<th>Count</th>
<th>Name</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>JOHNSON</td>
<td>8321</td>
</tr>
<tr>
<td>7</td>
<td>JOHNSON</td>
<td>497</td>
</tr>
<tr>
<td>3</td>
<td>497</td>
<td>265</td>
</tr>
</tbody>
</table>
Abstract Syntax Notation One (ASN-1)

- An ISO standard
- Essentially the C type system
- Canonical intermediate form
- Tagged
- Compiled or interpreted stubs
- BER: Basic Encoding Rules
  \((tag, length, value)\)

Network Data Representation (NDR)

- Defined by DCE
- Essentially the C type system
- Receiver-makes-right (architecture tag)
- Individual data items untagged
- Compiled stubs from IDL
- 4-byte architecture tag

- IntegerRep
  - 0 = big-endian
  - 1 = little-endian
- CharRep
  - 0 = ASCII
  - 1 = EBCDIC
- FloatRep
  - 0 = IEEE 754
  - 1 = VAX
  - 2 = Cray
  - 3 = IBM