Implementing Remote Procedure Calls

Birrell and Nelson

Reminders

- **538B project proposal drafts due tomorrow at 6PM Vancouver/PST time**
- Drafts are required
- Drafts are not marked
- I will try to get you feedback over the weekend

RPC

- RPC : preserve semantics of LPC (maybe?)
- Client invokes method, args are “magically” packed into msg, delivered to callee at server, the server unpacks the msg, and invokes the function, and sends the result back, client extracts result, and returns back to caller.
- Components (5): user, user stub, RPC runtime, server stub, server implementation.
    - Interface exported by the server/imported by the user
    - Interfaces create an illusion of an LPC (identical call infrastructure). The stubs do the REAL work of remote control transfer.

RPC

- *What jumped out for you about the paper?*
- **Binding** = naming + locating; name of interface = type + instance. Name resolution to resolve the interface to location. Grapevine DB ~ DNS.
- RPC: abstraction that hides the remoteness. e.g., blocking without any time-out! If you want timeouts, do what you would do for LPC calls.
- RPC : distributed programming with worrying about the network. It's a PL paper! Language + Usability primary concerns.
- Special casing the network to support RPC for efficiency (works because RPC is the only/rare special case).

RPC

- *What's not so great about RPC?*
- Implementation: broad assumptions about networks (private nets at Xerox). Optimizations are highly custom.
- Design: Scalability is mentioned, but not evaluated. One RPC call at a time (per process on a Dorado). No async RPC. Highly optimized for short calls (simple ACK strategy). May use many more packets than necessary; probe calls for maintenance of conversation (Ali: this is necessary! Noa: Probes not optimized).
- Grapevine dependency: Inter-dependency between DB and performance/behaviour of RPC. Failure of Grapevine?
- Security: They have it! But, few details. Leans *heavily* on Grapevine DB (key distribution/ACLs). Clocks important for security!? Poor clock resolution; 1s.
- Abstraction: *"exactly zero or once semantics as LPC". (Advantage of Xerox happy network).* Why are timeouts really that terrible? They are 'complex': you need a timer, you need a timeout value (transport logic) … a time-out value could be incorrect.
- Longevity of RPC: coupled to Ethernet.. that came out Xerox!? It builds on PUP, which looks very similar to TCP/IP.
- **Longevity of RPC: RPC uses functions as the modularity boundary for remote computation.**

RPC v. LPC

- RPC can't deal with *pointers* (as args/returns). The issue is lack of shared address space (resolved with distributed shared memory.. but this is used infrequently/few implementation exist).
    - Pointers a huge win: no data copying. Pointers are ~ data structures. I need some network-level representation for pointer-based structures.
    - Sharing is a problem: a single view of a data structure is violated across machines.
        - Modern solution: serialization (sort of helps with representation)
- Use existing PL abstractions to support the LPC view of RPC (Mesa PL has timeouts and threading library; exceptional control flow). Is RPC successful because modern language resemble Mesa?
- Exceptions are raised for network issues: leakage of remoteness (user code becomes aware of RPC != LPC).
- Only the exceptions defined in the stub interface can be raised (again, this is building on the Mesa PL abstractions, and relies on type-checking). Compiler can reason about distribution during typing checking phase (light weight reasoning).

RPC ~ LPC limitations

- *<u>True LPC semantics</u>: If I truly wanted LPC.. whenever RPC fails, I would crash the caller.*

- Why not? In a sense.. you can with their RPC system (depends on how you handle exception).

  - What happens if the callee divides by 0 and dies? Caller receives an exception.

- This is just a start: only concerns “control flow”. But same issue comes up for other PL concepts. Objects.. data structures…

Remoteness

- Semantics in RPC parallel distribution concerns
- I built a distributed system.. I actively decided not to build an LPC-based system. So, naturally you want to use distribution to your advantage.
    - Advantage: partial failures. *I want partial failures, instead of total failures.*
    - Advantage: performance — more machines crunching away at your problem.
- HPC (high performance computing) systems: typically focus on perf, typically have total failures (any component failures => brings down the entire system)
    - Why is this an okay trade-off for HPC? HPC is for compute. If you lose a portion of your calculation (for modelling protein folding). Then… it's sort of useless to continue.
- RPC .. not for HPC?

RPC : control flow transfer?

- Control flow at function modularity
- Can we go further: *continuation transfer*? e.g., serialize and transfer all the state necessary for an execution.
- This is a popular idea in mobile computing: link mobile+cloud into a single system (***cyber foraging by Satya***)

RPC

- *How is it done now?*
- Distributed DB for coordination? Consul, etcd,
- Reality of today != vision of the RPC paper. Perhaps the design/abstraction are very similar.
- “Public lookup service” doesn’t really exist. Modern systems are much more tightly coupled than what Xerox imagined.
- 1/2 of the vision exists (Finn): boiler plate/stub generation all exists. (Perhaps we lost our way in the 90s, but now we are back.. with things like Go!).

What other *abstractions* would you export from local context to a remote context?

- Functions (RPC)
- Continuations
- Shared data (CRDTs, shared memory)
- Objects (Emerald: distributed objects)
- Keys, authentication information
- Processes (“process migration” ~ Emerald does some of this)
- Orthogonal concern: security locally: address space, processes. Remote: *“proof carrying code”*

Next: Argus

- RPC simplifies distributed system construction, but can we go further?
- Can programming languages support distribution natively?
- What are the right PL abstractions for distributed systems?
- Argus is one of the first papers on these topics