15-712 Project: Active RDMA

Chris Fallin, Anshul Madan, Filipe Militão

1 Idea

Traditional distributed services have built upon RPC or RPC-like interfaces at the network level. In these systems, a server exports a static set of operations, typically at a high level. For example, a file server allows a client to open a file in one call, and read a block of data in the next. An alternate paradigm exists: RDMA (Remote Direct Memory Access), first proposed in [8], exposes data structures in the server's memory directly. Client applications can then, with knowledge of the appropriate structures, implement most intelligence on the client side and access shared data remotely. The advantage of this scheme is that it allows low-latency operation if the RDMA mechanism is implemented efficiently (i.e., in hardware or at the low level in a system software trap). It also allows more flexibility in client operations by allowing a finer granularity than a static high-level API, although this aspect was not evaluated in the original proposal. The disadvantage is that, for complex data-structure manipulations or pathological cases (e.g., a long linked list in memory), the cost of multiple round-trips over the network can be extreme compared to performing the computation locally at the server.

The proposal in [8] gets around this limitation by also allowing RPC in some cases (e.g., when their nameserver implementation cannot find an entry after a certain number of hash-table probes). However, we see a better way. In the spirit of Active Networks [7], we propose to allow clients to send code to the server to execute directly against the shared data structures. We call this **Active RDMA**. This functionality will allow for the *flexibility* and *client-driven intelligence* of RDMA while providing the *data locality* advantages of traditional server-side code.

This proposal differs in several ways from previous work and other alternatives:

- Active Networks are primarily proposed for customization at the network or routing layer: for example, upgrading TCP [6], or providing programmable packet-forwarding as in ANTS [9], PLANet [4] or FIRE [5].
- Active Disks [3, 1] execute code at a disk controller, increasing data locality for certain types of data preprocessing in the same way as Active RDMA will allow for storage applications. However, Active Disks are limited in scope to this data processing. In contrast, Active RDMA is a generic mechanism that not only will allow for storage applications but any distributed system that requires some shared state at a central server.
- Mobile code [2] is the name for a general research thrust of code migration over networks in order to build distributed systems. This is the most similar to our proposal. However, mobile code is generally a higher-level abstraction that does not provide direct access to a shared memory segment, for example, and provides more sandboxing and security that remove many of the potential performance benefits.

In order to narrow the scope of our proposal, we plan to focus specifically on environments in which security is not an explicit concern. This seems strange at first in the context of a network-centric proposal, but such environments do exist. For example, large computing clusters or datacenters with a single owner, in which all machines or network devices are trusted and performance is the paramount concern, are one possible application domain.

Furthermore, the assumption that security is externally provided (whole-cluster access control) or otherwise an orthogonal problem allows us to push Active RDMA's code execution to a lower level, bypassing layers for higher performance. We plan to define our interface to be sufficiently low-level that it could hypothetically execute directly on a network interface card, interacting with shared memory via DMA. This would allow for extremely low latency.

A secondary benefit to this, and one that we plan to explore, lies in the flexibility of executing arbitrary code segments. A client could dynamically generate this code; for example, a "grep"-like operation to search through some shared data structure could employ a custom-compiled inner loop. If implemented in the context of a distributed filesystem, this would combine the benefits of Active Disk-like data locality for filtering with the flexibility of custom code generation.

Our proposal thus encompasses these key points:

- Defining a low-level ISA (instruction-set architecture) for remote code
- Allowing clients to send remote code to a server to access shared memory directly
- Demonstrating the capabilities of this architecture to: (i) implement a real distributed system (specifically, a filesystem); (ii) provide unique advantages by allowing arbitrary remote code execution.

2 Work Plan

We will implement Active RDMA in several stages, detailed below:

2.1 Specifications

We must first specify the interface that the Active RDMA mechanism provides. This will encompass an ISA specification for the remote code as well as the user-level APIs to (i) export memory on the server side and (ii) send remote code to a server on the client side.

We plan to base our ISA specification on the low-level JVM bytecode format, primarily in order to take advantage of existing infrastructure and tools. There are several open-source JVMs from which we can borrow a JIT engine, for example Kaffe.

2.2 Execution Engine

In order to make the prototype development feasible, we will not initially pursue an optimized implementation of Active RDMA either in a network interface card or in low-level system software. Rather, we will prototype in userland. As mentioned above, we plan to base the execution engine on a JVM core.

2.3 Mobile Code Toolchain

In order to use mobile code, clients need to be able to write it efficiently. As a first pass, we can implement a simple assembler for the bytecode format. It might be possible to adapt a simplified compiler to this backend if we find ourselves with more time. However, we expect that for our demonstration applications, mobile code segments will either be relatively easy to construct by hand, or else will be generated dynamically.

2.4 Applications

We will demonstrate Active RDMA's capabilities with a distributed filesystem application. We can start by implementing basic functionality to access files in a simple backing store, provided by hooks on the server. Then, once that is complete, we can build a few specific demonstrations of the flexibility of Active RDMA's mobile code – at least the canonical example of remote-grep, and perhaps others.

2.5 Distributed Filesystem Evaluation

Given this distributed filesystem application, we will evaluate performance against NFS and AFS. We will evaluate raw I/O performance, and we will also develop benchmarks that take advantage of our unique mobile-code features to showcase the strengths of our implementation.

2.6 Additional Demos

If time allows, we will develop additional (small) demonstration applications that use mobile code in some way to implement some distributed functionality.

2.7 Goals

As a baseline, 75% goal, we hope to at least implement Active RDMA and get a simple filesystem to work on top of it.

Our 100% goal, and the one that we expect to achieve, is to build specific mobile-code filesystem applications such as remote-grep and demonstrate the strengths of our proposal.

If time allows and we wish to go much further, we will investigate (i) implementing further applications that do interesting things with mobile code, perhaps generating it dynamically, (ii) a more complete toolchain for mobile code, and (iii) kernel-level implementations of the Active RDMA execution engine.

3 Resources

We will require several network-connected machines to evaluate this proposal. In development, we can use either a set of virtual machines or our personal workstations. In the final evaluation, we may look into finding a more complete, larger test network to stress-test the implementation.

4 Intended Outline

- Introduction and RPC-RDMA Tradeoff
- Related Work
- Active RDMA
  - ISA specifications
  - several small examples of remote code segments (data structure manipulation)
- Active RDMA Filesystem (ARF)
  - Shared data structures
  - Server-side hooks
  - Client-server partitioning tradeoff
- Dynamic Mobile Code: Remote Grep
  - Mechanism: how to generate code
- Evaluation
- Future Work

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: programming model, algorithms and evaluation. *ASPLOS-8*, 1998.

[2] A. Carzaniga, G. P. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. *ICSE*, 1997.

[3] G. Gibson et al. A cost-effective, high-bandwidth storage architecture. *ASPLOS-8*, 1998.

[4] M. Hicks et al. PLANet: an active internetwork. *IEEE INFOCOM*, 1999.

[5] C. Partridge et al. FIRE: Flexible intra-as routing environment. *SIGCOMM*, 2001.

[6] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack. Upgrading transport protocols using untrusted mobile code. *SOSP-19*, 2003.

[7] D. L. Tennenhouse et al. A survey of active network research. *IEEE Communications Maganize*, 1997.

[8] C. Thekkath, H. M. Levy, and E. D. Lazowska. Separating data and control transfer in distributed operating systems. *ASPLOS-6*, 1994.

[9] D. Wetherall, J. V. Guttag, and D. L. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. *IEEE OPENARCH*, 1998.