Scalable and Fault-Tolerant Shared Memory

Dean E. Malone

September 1, 2017

Revision 1.5

Executive Summary

Can shared memory be made scalable and fault tolerant? This is an interesting notion. I believe the answer is yes but there are a few enormous challenges. In the circle of people I have worked with in this middleware space, much of what is needed already exists. We are literally twenty-and-goal away from scoring a touchdown; 80% of what is needed is already in production systems today.

We will explore what functionality is needed to support such a robust capability that works in a hybrid computing environment.

Table of Contents

Introduction 3
Here is What You Need 3
Final Thoughts 5

Introduction

When you think of scalability, you must consider both vertical and horizontal scalability. Vertical scalability is ability to increase the computing power of a particular server– for example, adding more CPUs, adding more processor cores to CPUs, or increasing memory. Horizontal scalability is adding more servers to increase aggregate computing capacity. Examples include using a front-end processor to take in requests on a well-known IP address; and using a cluster of NonStops across an EXPAND network.

All of this is well known architecture in terms of message-based client-server processing. But how do you scale vertically and horizontally in a distributed shared-memory environment to realize what Martin Fink labeled universal shared memory? That is what I have been thinking about. It is a new frontier.

It is also the next frontier of scalability and fault tolerance – shared-memory scalability over an infrastructure that can survive any single point of failure. This is an enormous challenge for architects to address. You don't see businesses building applications that can do this because there are no frameworks that make it easy to get it right the first time. Brokerage is the only industry using IB and RDMA at any significant scale, so there is still time to address this gap in the competitive landscape. I have written extensively on this topic and provided presentations to numerous parties. To read the latest iteration of my white paper on this topic, check out my web site to read They Don't Know What They Don't Know

http://www.caleb-ltd.com/Media/images/WhitePaperNSIMC.pdf

Here is What You Need

Let's "blue sky" visualize what such a truly capable framework should look like:

- Supports construction of massively scalable high-performance applications comprised of millions of devices that can access a huge pool of distributed shared memory resources (i.e. inter-process messages, semaphores and memory) without a-priori knowledge of where they reside. They obtain session bindings at runtime via a well-known hierarchy of DNS-like catalogs that map these resources from client applications to server(s) containing the resources they wish to access - an order queue, a status indication, notification of state changes, etc.
- Supports statically compiled programs, web pages, and Java interpreted programs that can all seamlessly access this framework.
- The framework behavior must be completely consistent on all platforms and across all network transport implementations.
- Supports 64 bit addressable shared memory.
- Access must be completely secure. Not only must the client provide a valid username and password over a secure session, but that client must also have appropriate read/write/read-write/delete authorization privileges – right down to a single semaphore, region of shared memory or message queue.
- Once logged in, unique security session handles are generated that can be included with subsequent operations and must have a finite time to live. They must be cached in protected shared memory that resides on each server containing resource entities, so that security authorizations and evidence of authentication can be quickly retrieved.
- Security profiles reside in LDAP compliant databases so the framework can integrate with and be maintained by enterprises.

- Utilizes RDMA over any supported network in compliance with InfiniBand Trade Association evolving standards.
- Resource access must be extremely fast. Utilizing a federation of servers to collectively replicate state (e.g. Hazelcast) is not an option. These replication frameworks are unworkable for frequently updated data elements because they propagate update broadcast storms. For transactions, the update operation must be synchronized with every participant.
- Can seamlessly handle release version differences (backwards compatibility) so that all client and server participants can interact without introducing compatibility failures.
- Shared memory must persist in exactly two servers – one primary and one backup – and all client APIs must have a QP (queue pair) session to each so that if the primary fails, the session is already established by which the client can recover. This is an important IDC AL4-compliance capability.
- Transactional semantics must be able to perform the following basic operations:
    - Begin Transaction.
    - Specify a list of shared-memory resources that will participate in the transaction.
    - Third-party operations will not able to access these resources until freed by a timeout, commit or back-out.
    - Save a "before image" of each participating resource and checkpoint each to the backup to facilitate fault-tolerant recovery.
    - Perform the operation(s) on the shared-memory resources.
    - Commit the transaction, freeing the mutexes, and cleaning up the before images.
    - Abort the transaction, restoring the resources to their "before image" state, free the before image resources, and release the mutexes.
    - Provide an external API that will allow existing transaction engines (e.g. TMF, Tuxedo, RDBMS engines, etc.) to integrate with this transactional framework. All transaction engines that have ever been built to date, commit and roll back **files**. I want to commit and roll back **memory resources**. The two will need to be integrated into a single transaction.
- Facilitates setting of priorities for different classes of customers and critical business functions that can "jump the queue."
- Entire distributed framework should be able to run on the following server platforms to be useful:
    - Smart phones
    - Linux
    - Solaris
    - AIX
    - Windows
    - HP-UX
    - NonStop
    - zOS
- Works *today* on legacy TCP/IP as well as RDMA networks like RoCE to *immediately* build and deploy something useful, and then seamlessly migrate to new RDMA-capable (e.g. RoCE and InfiniBand) networks as and when they become available.
- Supports containerized (e.g. Docker, etc.) deployment that can quickly spin up and wind down server instances, in support of public and private cloud deployments.

Final Thoughts

Are there any readers who would like to capitalize on this vision for your enterprise? If so, please reach out to me so we can further discuss what we presently have in place, and what our planned roadmap to all of these capabilities looks like. You will be pleasantly surprised to find out that tools exist which can do much of this, and with which to start your application construction right now. I can be reached at dean@caleb-ltd.com