Summary: A Note on Distributed Computing

Ingeborg Ødegård Oftedal, Hallvard Jore Christensen, Paul Philip Mitchell, Kyrre Laugerud Moe

Before we can dive into the paper we have to clarify its terminology. In the paper local computing refers to a system that handles one address space while distributed computing is defined as a system that handles more than one address space, this could be on different machines. The client of such a distributed system only knows about the interface it is interacting with, and nothing else about the recipient. These categories for local and distributed objects are not exhaustive.

2 The Vision of Unified Objects

For the programmers point of view the differences between distributed (two machines) and local objects (share address space), is not vast. The reason for this is that the object implementation is independent of the interface, and thus is hidden for other objects. For the programmer, the underlying mechanisms of for example making a method call are abstracted away, because the code for that call will look the same either way.

This is an extension of RPC (remote process call), which makes both cross address function call and local function call look the same. This allows marshaling of parameters and unmarshaling of the results by extending it to the object-oriented paradigm, which makes the use of objects a natural choice in distributed computing. This vision is based on that all invocations and calls from a system could be to objects on a distributed machin.

Local and remote function calls are obviously not the same, but with different implementation techniques you can join objects in an application and use the same technique for joining objects to another application. The techniques does not need to be implemented the same way for both objects.

The use of this model in application consist of three phases. The first phase is to create the application in a natural way, using defaults and the appropriate interfaces between objects. This will enforce the separation of the machine architecture from the potential need of performance tuning. The second phase is about performance tuning and could be necessary to analyze object communication patterns. This phase is also about choosing the correct interfaces for clients to export. The third phase is about testing and improving the interfaces for partial failures.

Whether a call is remote or local does not affect the correctness of the program, and is a justification of using this model. The location where the operation is carried out makes no difference for the correctness. A system like this could bring many benefits, specially for changes, maintenance and upgrades. This is because the implementation of objects can be changed, as long as the interfaces is constant.

There are three principles this vision is based on; an application has a natural object-oriented design, failure and performance issues should be dealt with in the implementation part, objects and their interfaces is context based. These principles are unfortunately false.

3 Deja Vu All Over Again

Through history many approaches has been done for merging the programming of remote and local computing has been done.

For the communication protocol, one path has been on integrating with the language model, another path on solving distributed computing problems that is inherited. The language path has gotten least furthest historically. This is because there has been few distributed applications, and the programming model interfaces was far away from the currently model in use. And this continued in a cycle. A reason for this could be the non sufficient models created for all parts, this has though gotten us closer to a unified model for local and distributed computing, as of now the object-oriented approach is.

The communication part of distributed computing is not a hard problem. The hard problems of distributed computing is how to handle partial failure, absence of a central resource manager, concurrency problems, ensuring good performance and differences in memory access for the local and remote entities.

There is alot that can be done in distributed computing, but it needs to be done on a higher level.

4. Local and Distributed Computing

Areas like latency, memory access, partial failure and concurrency makes up the major differences between local and distributed computing.

Latency might be the most obvious one with a difference in between four and five orders of magnitude on object invocation. Since the difference is so vast this turns into a major design area, that needs proper attention when designing an application. The vision of unified objects, can provide two-pronged answer. The first prong is to trust the evolution of hardware to steadily increase its speed, while the second prong is to admit the need for tools to track the pattern of communication between the objects. Weather or not this will be possible is difficult to say, and the challenges are many.

A more fundamental difference between local and remote computing concerns their pointers in regards to memory access. Local pointers are not valid in a remote address space. The programmer must be very aware of this. If not, a way to unify the programming model is to grant the underlying mechanism control of all memory access. This would create a barrier to the programmers though, and removes their ability to access pointers in their language. Total unity is still an issue though, when cross-addressing entities other than objects. Hence treating remote and local access the same way, is very dangerous and issues may arise. Therefore knowledge of the pointers is still important for programmers.

Both the local and distributed world have components prone to periodic failure. In distributed systems (in contradiction to local) the failure of components are independent. This creates an issue in locating and analyzing the failure, and insuring the state consistency of the

system which is not found in local computing. Therefore communication interfaces must be carefully designed to produce a consistent response and be able to reproduce its state. Using such interfaces to unify local and distributed systems we have to choose if the extra information from partial failures should be ignored, or redundant information should be added in case of local failures by treating them as remote.

In regards to concurrency the same dichotomy applies for a unified programming model. Either all objects gets added concurrency semantics, or we must just hope for the best when distributed.

5 The Myth of “Quality of Service”

It is how the object is implemented that decide of it deals with different kinds of situations like latency, memory access, partial failure and so on. If the system should be more reliable, it must be implemented in the interfaces for that system, and is described as the “quality of services”. The interfaces can though not ensure the robustness of all the components in a system. In many applications robustness can only be assured at the protocol/interface level. The same yields for performance of a system, where several operations only is possible at the interface level.

6 Lessons from NFS

The ignoring of the differences between the interface level at distributed and local computing, could cause consequences, NFS is an example of this. But before NFS, one of these calls returned a rare error status, indicating catastrophic failures. Then NFS allowed partial failure in a file system. They did it by using two modes who dealt with an unavailable file server, hard mounting and soft mounting. What they didn't do, was changing the interface of the file system to deal with both local and distributed file access. The NFS protocol is stateless and pure client-server. This indicates failure for only three parties, server, client or the network. Because of this limitation, the handling of failure is simpler. NFSs constraints on reliability and robustness lays thus not upon the how the parts of the system is implemented. The problem is that they did not change the interface to fit with distributed computing. Even though, NFS application is the distributed application that has been most successful. But because of its lack of scalability and reliability, it has been adequate for a limited number of machines.

7 Taking the Difference Seriously

All the differences discussed make merging the computational models a less than good idea, to instead accept that the differences are irreconcilable and have them in mind in every stage of the implementation would be the best thing to do. Though interfaces are not the answer to unify local and distributed computing, they are very useful tools in controlling the differences if they are properly reflected in the implementation. Engineers will still have to know if they are sending messages to local or remote objects, and alter the technique used accordingly

8 A Middle ground

As we mentioned in the beginning the distinction of local and distributed objects are not exhaustive, in particular in regards to a third category of objects that exists in different address spaces but is also in the same machine. They have characteristics of both local and distributed objects, and should be treated separately. These objects are managed by a single resource manager, this causes partial failures and the following indeterminacy to be avoided, and latency can be reduced to acceptable levels. Still models are necessary for memory access and concurrency, but this may not affect the construction of interfaces that much.