CS290B: Java-centric cluster and concurrent computing
Marianne Melhoos, Johannes Lier, John-Olav Storvold, Julie Rekdal
UCSB, Spring 2015

Summary: A Note on Distributed Computing

The paper states that the common vision about objects, that they are essentially the same kind of entity, are wrong, and that there are fundamental differences between the interactions of distributed objects and the interactions of non-distributed objects.

The vision of unified objects are the vision of treating distributed objects and objects on a local address the same way. The general view on distributed object-oriented computing is that there is no significant difference between objects on a local address space and objects that are present in several systems possibly with different architectures. This is because the underlying mechanism for method calls are hidden from the programmer. A system with unified objects are not yet plausible.

Development of communications protocol has usually followed two paths: Emphasizing integration with the current language model and emphasizing the problem solving inherent in distributed computing. The language approach has only resulted in a cycle where they are back at start. The programming distributed applications are not the same as non-distributed applications which lead to failure at unification. The hard problems in distributed programming concern dealing with partial failure and the lack of a central resource manager, ensuring adequate performance and dealing with problems of concurrency, and the difference in memory access paradigms between local and distributed entities.

The main differences between local and distributed computing concerns latency, memory access, and partial failure and concurrency. Concerning latency, the disparity in efficiency that is normally seen as the most important difference between local and distributed computing.
Memory access: Pointers in a local address space are not valid in another address space. To solve this problem, either the memory access must be controlled by the underlying system or the programmer must be aware of the local and remote access.
Partial failure and concurrency: Partial failure is total or detectable in local computing while one component can fail while the other continues and you cannot determine what component has failed.

Robustness is not a purely based on the implementation of the interfaces. The implementation of an object affects the quality, or reliability, scalability or performance, of the interface. If you fix one problem it may lead to another problem. Robustness of the individual components has some effect on the robustness of the overall systems and but it is not the sole factor determining system robustness.

NFS opened the door to partial failure within a file system. It has soft and hard mounting to deal with an inaccessible file server. It is stateless, so it the server does not care what happens to the client. The problem is that interface was designed for non-distributed computing where partial failure was not possible.

Merging of the computational models by making local computing follow the model of distributed computing would require major changes in implementation languages and make local computing more

complex than necessary. Merging the other way will lead to systems that are unreliable and incapable of scaling. Instead of merging the engineers need to know when it is appropriate to use each kind of object.