Distributed Sets of Objects
László Böszörményi
Institut für Informatik, Universität Klagenfurt
Universitätsstr. 65 -67, A - 9020 Klagenfurt
e-mail: laszlo@ifi.uni-klu.ac.at  http://www.ifi.uni-klu.ac.at/staff/stafftab.html

The basic problem of object-based distributed systems is that the abstraction level of discussion is too low.

We generally speak about object distribution, which is not really relevant for most applications. Applications are generally not interested on having arbitrarily distributed objects, rather on aggregates of objects that are themselves distributed.

Maybe the most important difference between a desk calculator of the fourties and a modern (von Neumann or Zuse) computer is the capability of the latter to process aggregates of data (actually arrays of numbers), instead of processing data individually. In a similar way, the most important difference between a computer network and a true distributed system could be seen (beyond all the usual transparency requirements) in the capability of the latter to process aggregates of distributed objects in a transparent way. Finding the proper aggregate is the only way to free the application programmer from struggling with the location of individual objects. Rather, he/she gets a collection of objects, which is automatically distributed corresponding to the semantics of the application on the one side and to the underlying architecture on the other side.

Claiming pre-defined distributed collections instead of freely distributed objects imposes surely a kind of restriction. But restriction is often exactly the key for making new technology available to a broad user community. Such an example was structured programming, restricting flexibility in programming. A maybe still better example is SQL, restricting even the power of the access to data - the key for its unrestricted success.

In the area of classical parallel processing (scientific computing) this has always been known. Scientific programming concentrates on parallel arrays of numbers [Zim92, PT91]. This approach has its great advantages in semi-automatic parallelization of programs processing large matrices, it has, however, its limits as well, and can surely not serve as a basis for distributed object systems.

We propose as such an aggregate, the concept of distributed, polymorphic and persistent sets of objects. This kind of aggregate has the following advantages:

1. Sets are well-understood, relying on a sound mathematical theory.
2. A set is per definition an unordered collection, therefore, it can be processed in parallel most naturally.
3. Data are retrieved from a set based on their values (instead of computed positions), which is the usual view of most applications coping with a large amount of data (e.g. of most database query languages).
4. There are a number of reasons to combine distribution and persistence, being related in a most inherent way: Distribution is an extension of scope in space, persistence is the same in time. Most applications, requiring distribution of objects, also need persistence of the same. Moreover, some technical problems are very similar. For example, objects needs an identification which is independent from both space and time. A framework that considers
both of these aspects can provide its user a view on object aggregates, which is transparent both to location and duration (i.e. the user neither have to struggle with locating his/her objects nor with saving and reloading them).

5. Polymorphic sets can store objects belonging to an object type hierarchy. With the help of polymorphic sets we can easily build hierarchical subclasses on the extensional level [BEW94].

**M3Set and PPOST**

Based on these considerations we have designed and implemented a system that fulfills the above requirements [BE94, BE96, BEW94]. It supports the concept of distributed and persistent sets on two basic levels: on the language and on the run-time level.

In a recent paper [DNSV96] DeWitt et al. introduce a similar approach, called ParaSet, based on the SHORE OODBMS and Parallel Sets of Kilian [Kil92]. They show with the help of the OO7 benchmarks that the approach results in excellent traversals. They offer a C++ library, but no direct language support yet (it is planned).

**Language level (M3Set)**

Distributed and persistent sets of objects are provided as "first-class-citizens", thus allowing the compiler to guarantee strong type checking and optimization of set expressions - realized as an extension of a clean, object-oriented language, Modula-3 [Nel91]. M3Set provides a declarative, very powerful set-expression (a kind of generalized select), which suits ideally for optimization and parallelization. Thus, for relatively low effort (compared to nested-loop parallelization of typical parallel programming languages) high power can be achieved. Common subexpressions can be optimized by the compiler statically. Cost-based optimization and parallelization can be done at run-time, dynamically. The entire process of optimization and parallelization is integral part of the language system. The language allows very different implementations. Any combination of persistent and volatile, parallel and sequential, optimized and unoptimized implementations is possible. This is eased by the fact that a great part of the implementation is located outside the compiler with the help of predefined interfaces.

Set variables can be declared as being distributed and/or persistent. In the case of distributed sets a distribution strategy can be specified, such as range or hash distribution. The hash or range function must be supplied by the user. The actual distribution is made, however, automatically.

**Run-time level (PPOST)**

Although on the language level several implementations are possible, the most relevant implementation is based on PPOST (a Parallel Persistent Object STore) [BEW94]. PPOST implements a memory-resident, parallel and persistent object store. Different object-oriented database management systems might be built on top of PPOST. The term memory-resident (or main memory based) means that the primary storage device is main memory. Persistence is guaranteed automatically by managing secondary and stable storage devices.
Main memory-resident databases [AB92, GMS92] are often considered obsolete, because of their limited capacity of main memory and their inability to scale up with growing needs. This objection is not true any more, if a memory-resident database is implemented on a distributed architecture, which not only can incorporate substantially more storage capacity (possibly several Gigabytes or even more), but scales even better than disc-resident databases. Adding new nodes add not only more storage capacity, but corresponding processing power as well.

Another advantage of using parallelism in a memory-resident database is, that logging, checkpointing and archiving can be made in parallel with ordinary transactions serving the users.

Theoretical investigations and measurements show [BEW94] that such an architecture has a high practical relevance for applications, which have to process with high speed a large number of (mainly persistent) objects.

The following figure shows the architecture of PPOST. The objects contained in a distributed set are located on the nodes of the object-store (horizontal parallelism). Transaction starts and commitments are handled by the transaction node. Persistence is realized by the log, checkpoint and archive nodes (vertical parallelism). The log node records all changes in (fast) sequential log files. After this operation the corresponding transaction can be committed. All other background operations on disks or tapes are executed in parallel to normal transaction processing.
Conclusion

We claim that distributed object-oriented systems must provide a higher level of abstraction to their users, than usually provided. Especially, it is necessary to provide application-oriented, intelligent aggregates of objects with transparent distribution of their elements. Beside that, it seems to be not only reasonable, but also relatively easy to connect persistence with distribution. A system, offering distributed and persistent sets of objects on the level of a clean, type safe programming language, can provide the necessary level of abstraction, many applications need. The user of such a system gets distribution and persistence in the same "natural" way, as users of traditional systems get volatile arrays of numbers or classes of objects.

References


