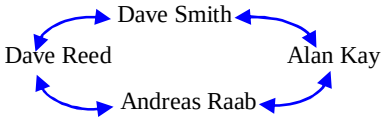
Croquet: New Models For Massively Scalable Sharable Computation



10 Second Version

Math wins! The properties and restrictions of mathematics that allow powerful nontrivial reasoning also helps tremendously when adapted to computing processes and structures. In this case, it is finding a mathematical model of time, and behavioral-change through time, that provides great benefits in massively scalable real-time coordination of large complex computing processes.

What is the Croquet Project?

Croquet is an ambitious attempt to create a:

- *"simulation style" dynamic-object computing model* and computers to run it (both virtual and real)
- *operating and networking system* that automatically distributes and coordinates computations over the Internet
- *programming model* for all levels of programmers from children and novices to experts
- *set of user interface approaches* for allowing all levels of users to navigate, use, and author "situated objects"

What Does This Mean in Practical Terms?

- the computing model is considerably more simple, safe and powerful than current day practice.
    - this means that bigger projects can be safely done by fewer computerists at much lower cost for both development and life-cycle maintenance.
- The automatic distribution of computations ensures atomic transactions on all coordinated objects in real-time in the presence of errors and delays.
    - this means that it is possible to easily build really inexpensive massively scalable peer-peer systems of all kinds using only the end-user's computers, that act as though they are immensely powerful and speedy central shared resources with backup and undos.
- The new programming model allows objects to be composed "sideways" and scripted "by tiles" without the need for complex inheritance schemes and extensive memory of the scripting system.
    - this means that programming can be an inherent part of every object's description for all users, and introduces a new kind of computer literacy into the world
- The new approach to UI in this system recognizes that "everything is UI" for most users, including expert professionals, and tries to supply an underlying uniform approach and philosophy for the many custom designs that will be required in a real-world system.
    - this means that much less detailed knowledge is required to learn all aspects of user interface of the system, and the uniform approach should make a large difference in its usability and approachability.

In Prose

Croquet's treatment of distributed computation assumes a truly large scale distributed computing platform, consisting of heterogeneous computing devices distributed throughout a planet-scale communications network. Applications are expected to span machines and involve many users. In contrast with the more traditional architectures we grew up with, Croquet incorporates *replication of computation* (both objects and activity), and the idea of *active shared subspaces* in its basic interpreter model. More traditional distributed systems replicate data, but try very hard not to replicate computation. But, it is often easier and more efficient to *send the computation to the data*, rather than the other way round. Consequently, Croquet is defined so that *replication of computations is just as easy as replication of data*.

See http://atsosxdev.doit.wisc.edu/croquet2/ for more comprehensive information about the Croquet project, including many screenshots.

The Context and the Issues

We are now computing and communicating via a world-wide network of hundreds of millions of nodes that will eventually grow to include all the people and devices on the planet (and if Vint Cerf has his way, the solar system).

We are also starting to seriously employ *multiprocess* (and especially *multiprocessor*) computing to get around single-processor bottlenecks.

Both of these areas use communications systems with delays that are long enough to require various kinds of replications of state to make the computing and messaging more efficient. And there are always errors …

We'd Like …

- the world-wide complex of networked computers to function like an infinitely fast and capacious error-free computer with no communications delays
- e.g. to provide wide simultaneous realtime safe access and authoring to very large "object-bases" running from very "data-like" holdings such as banking and other financial "data", to immersive virtual environments of many kinds, to complex simulations involving thousands of computing elements. And we'd like the object model to unify all of these areas.
- this to be done in a way that scales no worse than linearly: ideally, we'd like to only have to use the computer that each end-user employs to get on the network and use it
- to have this work on every different kind of computer pretty much regardless of speed and type
- to allow all kinds of authoring, including end-user authoring, to be done safely at all times
- etc. etc.

The Internet Solves a Massive N^2 Problem

The idea of the Internet was to allow any node to communicate with any other node regardless of the total number of nodes. The model is that of a cross-point switch, which scales as N^2. Since the Internet was to be "intergalactic" in size and scaling, the solution was to use peer-peer packet-switching-and-routing so that, in principle, only the end-node machines need to be used.[1]

Shared Computing is a Supermassive 2^N Problem

If there are 100 users on a net, $100^2 = 10000$, but the number of possible subgroups is $\sim 2^{100} = \sim 10^{60}$. This is close to the number of electrons in the universe, so even for a small network with not all subgroups being formed, we generally do not want to try solving the sharing problem using centralized resources!

Since we want to share in real-time (or "node-time" i.e., that is commensurate with the time passage regime of the node), we have to figure out how to trade off the current large computing and storage capacities of computers (even consumer computers) that are hooked together with slowish networks to make a new kind of real-time peer-peer model that scales no worse than linearly.

Simulating Time

We have at least three kinds of time that we have to deal with:

- There is a "real-time" that can be fairly closely agreed on via time bases on the Internet (note however that actual *exact* synchronization is not possible, just "close synchronization").
- Then there is "node-time", the local "hardware time" of each machine involved. We can think of this as ticks/second, and it could easily vary by a factor of ten or more, depending on the speed of each machine.

[1] In practice, there is more structure, but not exponentially more, and the base principle is pure and simple.

- Then there is a pseudo-time for each object and for systems of objects. This is the only time that can be *exactly* agreed on because we can *simulate* just the time-regime that will be most helpful.

Race conditions on state can and often do happen in "normal computing in real-time", but it is possible to avoid them with a different computing model that uses a simulated time regime to control access.

To take just one way of doing this: we can decide to represent any interesting object as its *history of stable states*. This means that *any viewing* (including by the object itself) can only see a stable state, and any computing that takes place *in* the object to assist viewing has to be a pure function. This means that even when the object tries to update itself to the next stable state, it is restricted to use only its previous stable state as a source for values. This is like a two-phase clock in hardware design. (This is a very good thing, and it can be made to look very much like ordinary programming.)

Since *real objects*[2] are by their definition protected, and must receive and adjudicate messages-events to initiate processes that might change their state, we can also keep a *message-event history* that keeps track of the ordering of the messages as they are issued and received.

What is a DistOb (Distributed Object)?

One way to distribute an object to many machines would be to have a kind of proxy object that simply routes all messages to and fro a master object that exists on one machine. The master can take care of sorting out its message-event stream, can update as it chooses, and then send appropriate replies back to the requestor. Note that this is not a very scalable solution, as our master object is acting as a central server. Also, there could be quite a bit of state to send to each proxy object, thus taxing the generally low bandwidth of the WAN.

However, there are certain extreme cases where this could be a good idea: e.g. if this object represents an unbelievably massive computation – such as a billion point 2000 params/point hydrodynamics simulation - that can only reside on a supercomputer at present.[3]

A more general way to think about this is to have an object actually be copies of itself (replicas), each one on a user's machine, and to set up processes that will keep the replicas coordinated "more-or-less" in real-time. These processes will ensure that there cannot be any cases where any of the replicas can give a different answer from the others. In other words, we want something like a distributed banking transaction in which all meaningful actions are atomic, that is: not visible to viewing processes regardless of situation or possible errors. And we want these atomic transactions to be on all *computations* that matter, not just on data-structures.

A Croquet Example

Let's take a simple case from Croquet. We have a number of users anywhere on the Earth using the Internet, each in the "same" (replicated) shared space looking at the scene from the point of view of their avatars. One of the users clicks and drags on a hyper-portal to lift it. All the other users see this happen, and they can also grab at the hyper-portal and do things with it "simultaneously".

What is actually going on? Here's a somewhat simplified walkthrough for what will happen without any error conditions.

User A clicks on the hyper-portal P on his screen.

Actually, before anyone, including A, sees anything, an event (*portal P has been clicked on by A*) gets broadcast to all the users in this space, directed at the DistOb-P. We want to think of DistOb-P as all of its replicas P_1, P_2, P_3, P_4, P_5, etc. They all get the message-event at somewhat different real-times, but the message-event itself is labeled and ordered, as is their state-history.

---

2 As opposed to so-called "objects" in many so-called "object languages" which really are data-structures with procedural attachment and don't really encapsulate and use messages for communication.

3 But note that supercomputer solutions are now more and more massively parallel themselves, and might very well have to use more general schemes to deal with their own models ….

The replicas start independently computing the future that was implied by this event. They are given a *deadline in approximate real-time* to compute this future.

Now there's a tricky part whose details I'm going to skip over for the moment. The short version is that we have to ultimately have a distributed atomic transaction on our objects. In other words, at some point the system of replica objects has to commit to the same state everywhere in the same pseudo-time and as close to possible in the same real-time. If things go wrong — and there are many kinds of things that can go wrong in a highly distributed system of machines and networks of different speeds and reliability — we have to make sure that when the dust clears either the entire new version got computed, committed to and labeled, or no part of that transaction happened at all. (Like a banking transaction, we can't have money being moved from one account to another be disturbed by any kind of error or crash. The source has to decrement and the destination has to increment, or neither.)

Once they all commit to the new versions of their replica objects, they can generate a new display on the screen from the new objects. A very important thing to note here, is that the frame-rate that each user's computer can generate can be very different because the frame-rate has nothing to do with the simulation processes underneath. They have to track transactionally, but the displays are just viewing processes that view as much as the local machine capacity permits. This is pretty darn cool!

There are some other really nice things that happen naturally with this model of computation. Suppose we set an object in motion. What does this mean? In this scheme, it means that we have a model of what that motion is supposed to be given in real-time units (such as distance or revolutions per second).

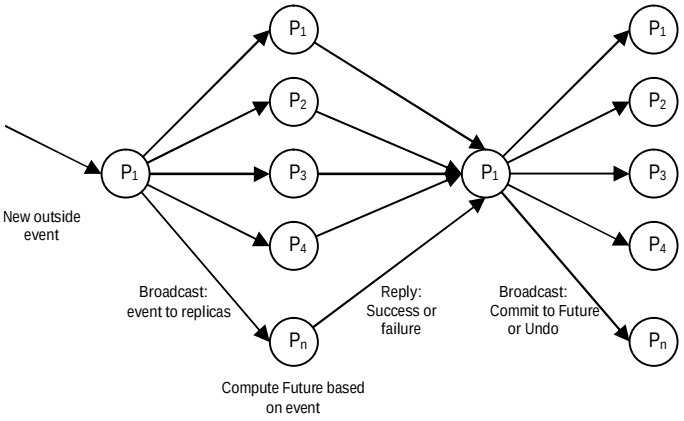
Let's take the simple case first, that the motion is uniform. This means that the replica can "take care of themselves" Regardless of the frame rate and hardware time of a local machine, each replica can compute its future positions quite accurately in a very interesting way that resembles a continuous function of time. E.g. when each machine decides it can compute a frame, it can ask the replica "where are you now?" and each may come up with a different answer *because the display question is a real-time question and will be different for each replica on different speed machines.* Yet, each replica has the same meaning for its behavior and state.

The seemingly more complicated case of an object changing its state in a non-uniform way over time is quite similar and also makes use of the inherent use of relating behaviors to a continuous model of time flow.

This is a good place to ponder the implications of the above scheme, and to realize that it is a very general solution for coordination of very large systems.

A Few More Details

How do the objects know whether they should commit or not? They have to coordinate, and the coordination has to be done in a way that is no worse than linear. Dave Reed's solution to this is to let the initiating object that did the original broadcast decide. So the general progression of events is roughly as follows:



It is only after an object has committed to a new state that viewing processes (such as computing a display) are allowed (because viewing processes are only allowed on stable versions of the object).

So we can see that "real-time" here to a user that initiates an action depends on the round-trip time to the farthest away (in Internet temporal terms) user that is in the space (i.e. the *report back success or failure*, and then the broadcast of *commit or undo*. Even though many frames per second can be generated by an end-user machine, a frame that shows some new action from an end-user event will be delayed by the round trip time.

But this is pretty good these days. In tests that we've done, the typical round-trip times in the US are about 30 to 60ms, and e.g. from Los Angeles to Magdeburg, Germany the round-trip times are typically around 100ms. The latter just exceeds the 12-frames/second guideline that E&S found was critical for jet fighter simulations. In sum, this scheme works very well over virtually all of the Internet using only the computers that the end-users already have.

A Little More About Croquet

The real-time 3D Croquet demo presents an extreme but very important case in coordinating state and process over any network or system, especially a worldwide network and system: that of a real-time 3D shared environment for shared play, work, and state.

The current design has been set up to test many concepts: ranging from the lowest level OS and metaobject architectures, to how all levels of scripting will be done, to how user interface conventions for navigation and authoring can be the most simple and powerful.

Analogies to the Web

For example, one experiment is to see if a sweeping analogy to the WWW works for end-users, both artistically and pragmatically. The current large-scale environment for Croquet is in terms of extensive 3D "worlds" (called "spaces") each of which is analogous to a single web-page. The Croquet "portals" are analogous to visual hyperlinks on the web that connect web-pages. There are also analogies to "bookmarks" (called "snapshots") that remember a space and a position in the space.

If a user enters a portal to a space that is not locally cached, the object replicas for that space will be fetched, just as with the resources of an uncached web-page. By far the largest amount of data that has to be fetched are the textures. These are sent as jpegs, and are thus quite analogous to the jpeg images that are part of the resources of an image rich web-page.

Progressive Updating

Croquet caches spaces for quite a bit longer than most web-browsers, in part because there is now plenty of room on most hard-disks to do so.

This means that there will be a rather subtle set of processes that will ensure that cached spaces are up to date when they are finally entered by the enduser.

This also means that error conditions, drop-out, etc., will be updated as gracefully as possible in a way that balances the tradeoffs of reloading, vs. progressive updating.

Userless Peers

Croquet is a peer-peer system, and only requires as a minimum the computers of each user in order to create a virtual shared environment. One of the things that is nice about peer-peer solutions is that they can also be used in a wide variety of augmented configurations. We have already mentioned the use of a replica as a proxy to a supercomputer. Another interesting use of the peer-peer architecture is to create "userless peers" that act as Bishop Berkeley's observers in the forest, e.g. in the proposed 200,000 user system for the Universities of Wisconsin and Minnesota, there will be a number of "userless peer" machines that will effectively carry out backup and other "continuity processes" by gradually caching large parts of the system.

To Find Out More About Croquet

Besides the Viewpoints Research/HP group of researchers, the two largest developers and users of Croquet currently are a combined group from the University of Wisconsin and the University of Minnesota under the joint leadership of Julian Lombardi and Mark McCahill. This is a very ambitious large effort to make a virtual university for more than 200,000 students. See http://atsosxdev.doit.wisc.edu/croquet2/ for more comprehensive information about the Croquet project, including many screenshots.

There is now a Croquet “Board of Directors” that is comprised of the most active participants in Croquet development. Currently this includes the four original Croquet designers — Dave Smith, Dave Reed, Andreas Raab, and Alan Kay — and the two newest members: Julian Lombardi and Mark McCahill.

Another large group from one of the Japanese national research labs is starting to use Croquet for a major project and is starting to donate funding (to the Wisc/Minn group with whom they are starting to collaborate). Their research has a number of facets, but is mainly devoted at the present time to making large shared environments for holding computer models of ancient archeological artifacts.