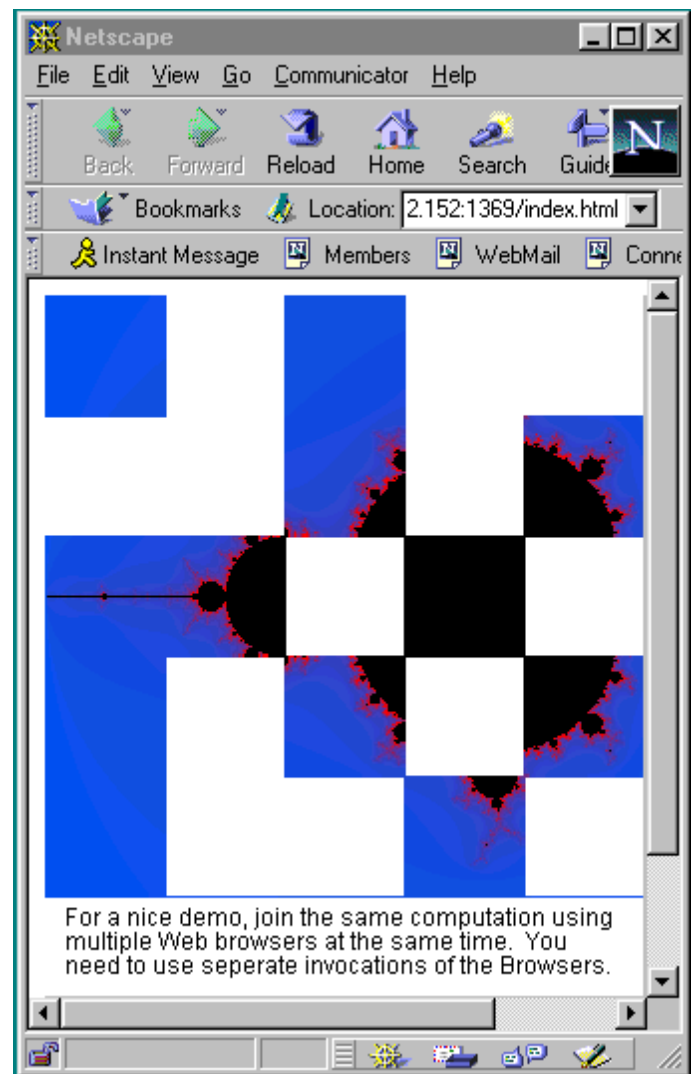
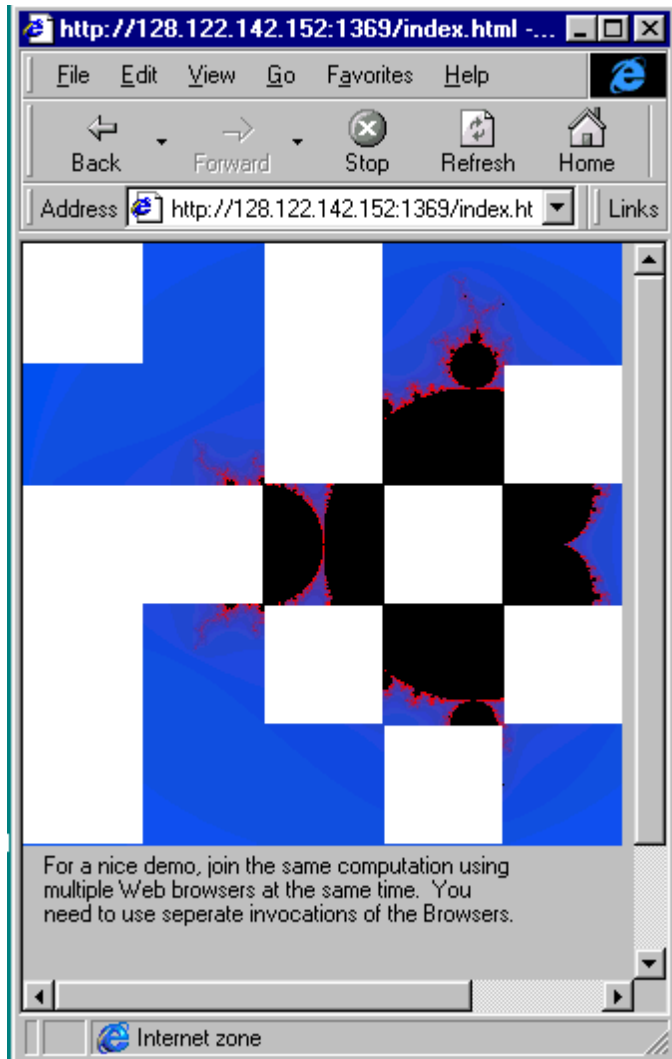


Charlotte

Metacomputing on the Web



Goals and Functionality

Charlotte is the first integrated environment for parallel computing on the Web. Now, using a Java-capable browser, *any* machine on the Web can volunteer to help an ongoing computation, with the Web potentially becoming a gigantic parallel metacomputer.

To use Charlotte, the application programmer writes software for a perfect shared memory virtual machine. Charlotte's runtime system implements this virtual machine out of the available volunteers, providing load balancing and fault masking.

An application program is a pure Java pro-

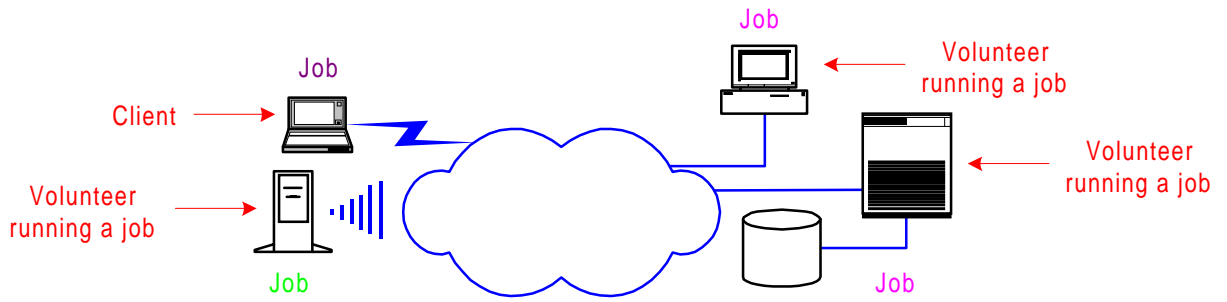
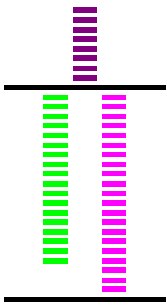
gram with embedded constructs specifying parallel steps. A parallel step consists of concurrent jobs, executable as applets.

A parallel computation is started by the user's machine. Machines on the Web can volunteer to execute a concurrent job, with the applet and the data downloaded through browsers, as needed.

As Charlotte targets the Web, the computation transparently crosses administrative domains, with the matching of volunteers to computations accomplished through a scalable and fault-tolerant directory service. The overall security is at the level of Java.

Some Implementation Details

In order to implement the predictable, perfect virtual machine on the unpredictable, dynamically changing Web, Charlotte uses two integrated techniques, eager scheduling and two-



The machines on the web volunteer to run Java applets executing concurrent jobs in a parallel step—part of the client's computation.

Eager scheduling and *two-phase idempotent execution strategy* provide transparent fault masking and load balancing.

Correctness is assured despite potential multiple executions of a job—such execution of a “purple” job on two volunteers is shown above.

A Charlotte program and its execution on the Web

phase idempotent execution strategy (TIES).

Employing these techniques, an unfinished job can be assigned/reassigned, allowing multiple and partial executions, while maintaining correct once-only semantics. The performance of the volunteer machine is not monitored. Nevertheless, the machines are automatically shifted to increase performance or mask slowdowns and crashes.

The runtime system is optimized to improve performance. A key technique of particular benefit to application software programmers is *dynamic granularity management*. Even when the programmer has written a fine-grained program, the runtime system will execute in an adaptive coarse-grained manner. Programming and execution parallelisms are de-

coupled.

An example shows two browsers running a worker applet contributing to the computation of a Mandelbrot set. Notice how the total work is divided among the two browsers.

Bibliography (unless indicated otherwise, available from MILAN web sites at NYU and/or ASU)

A. Baratloo, H. Karl, M. Karaul, Z. Kedem. An infrastructure for networking computing with Java applets.

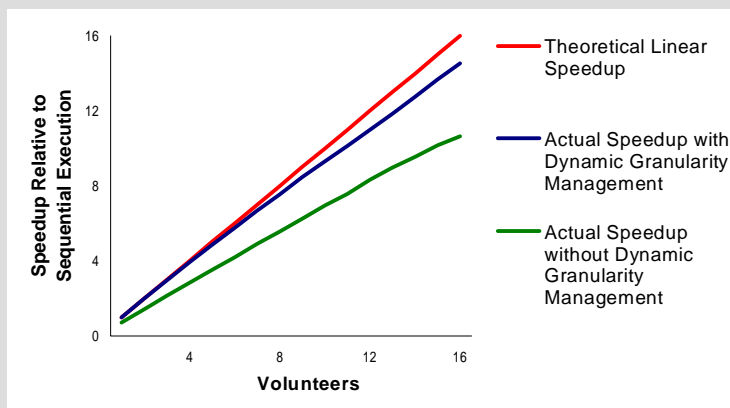
A. Baratloo, M. Karaul, Z. Kedem, P. Wyckoff. Charlotte: Metacomputing on the Web.

W. Gibbs. World wide widgets. *Scientific American*, May 1997. (Describes several metacomputing projects, including Charlotte.) Available at: <http://www.sciam.com/0597issue/0597cyber.html>.

H. Karl. Bridging the gap between Shared Memory and Message Passing.

Program fine-grained / Execute coarse-grained

Dynamic granularity management, one of several performance-enhancing techniques, transparently executes fine-grained computations in a coarse-grained manner, adapting the execution to the runtime characteristics of the volunteer machines. The example shows an actual performance of the Ising model computation.



**MILAN is a joint project of
New York University
Arizona State University**

Zvi M. Kedem
New York University
+1 212 998 3101 (phone)
+1 212 477 3265 (fax)
kedem@cs.nyu.edu
<http://www.cs.nyu.edu/milan>

Partha Dasgupta
Arizona State University
+1 602 965 5583 (phone)
+1 602 965 2751 (fax)
partha@asu.edu
<http://milan.eas.asu.edu>