

Metacomputing on the Web: The Charlotte Project

Distributed Systems Laboratory at New York University
www.cs.nyu.edu/milan/charlotte

Parallel computing is faced with a strange dichotomy. On one hand, the World Wide Web connects millions of mostly idle machines creating an attractive resource for parallel computing. On the other hand, the research aimed at utilizing local area networks as metacomputers does not easily extend to the Web. The *Charlotte* project aims to remedy this.

Utilizing the Web as a metacomputing resource introduces a set of difficulties and problems:

Programming Model: The Web is an inherently dynamic environment exhibiting unpredictable behavior. The programming model should be decoupled from the complexities of the execution environment by providing a virtual machine abstraction with uniform and predictable semantics. The execution environment needs to implement the virtual machine.

Heterogeneity and Portability: The Web encompasses different types of hardware, operating systems, and networks which make support for heterogeneity and portability an imperative.

Security and Accessibility: To allow any machine on the Web to contribute to any computation on the Web, security measures must be provided to protect machines against potentially harmful programs.

Lack of Common Resources: The Web is comprised of many administrative domains and invalidates many of the assumptions made for networks of workstations (e.g., no shared file system, no one has accounts on all machines, not homogeneous). A solution for the Web cannot rely on any of these assumptions.

A comprehensive solution for using the Web as a metacomputing resource requires that all of these issues be resolved, which Charlotte does.

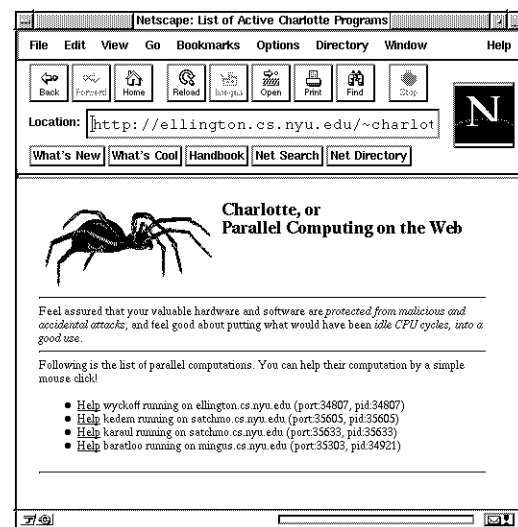


Figure 1: Several Charlotte applications looking for CPU donations

Charlotte is based on provably correct methods for executing parallel computations on imperfect machines. These methods were validated in the *Calyso* system for homogeneous systems. Charlotte builds on this and other complementary research efforts. The key properties of Charlotte are:

- Charlotte is the *first programming environment for parallel computing on the Web using a secure language*. Charlotte is built on top of Java without relying on any native code and provides the same level of security, heterogeneity, and portability as Java.
- Charlotte is the first environment that allows *any machine on the Web to participate in any*

ongoing (Charlotte) computation. Two key factors make this possible. First, programs are transmitted to participating machines at runtime—a shared file system or a local copy of the program is not needed. Second, a user does not need an account to utilize a machine on the Web—the decision to involve a machine in a computation is made by the owner of that machine or a process acting on behalf of the owner.

- A novel technique for providing a *distributed shared memory abstraction*. This is realized at the programming language level without relying on operating system or compiler support.
- Charlotte employs two integrated techniques, *eager scheduling* and *two-phase idempotent execution strategy*, for load balancing and fault tolerance in order to deal with the dynamics of the Web while providing the semantics of a reliable virtual machine.

A Charlotte program is a standard Java program with parallel steps. Computationally intensive parts are performed in parallel steps by one or more parallel *routines*. A routine is analogous to a standard Java thread, except for its capability to execute remotely.

The data is logically partitioned into *private* and *shared* segments. Private data is local to a routine, whereas shared data is distributed and can be accessed on remote machines. Consistency and coherence of shared data is maintained transparently by the runtime system. The memory semantics used in Charlotte is *Concurrent Read and Unique Write*.

A Charlotte program runs as a regular stand-alone Java application. When started, it creates an entry in a certain Web page (see Fig. 1). By clicking on such an entry, an applet is downloaded, allowing everyone with a Java-capable browser to donate CPU-cycles. Charlotte’s run-time system can tolerate any number of applet crashes and guarantees that slow worker machines do not slow down the overall execution of the distributed application.

As an example, Fig. 2 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.

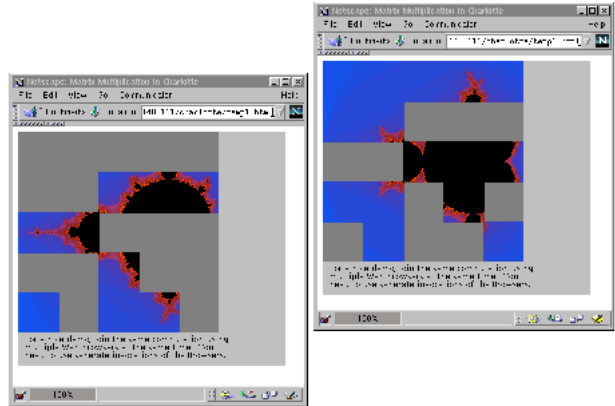


Figure 2: Two Charlotte Applets computing the Mandelbrot set

For detailed information on Charlotte, see www.cs.nyu.edu/milan/charlotte and [1].

References

- [1] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the Web. In *Proc. of the 9th Intl. Conf. on Parallel and Distributed Computing Systems*, 1996.