Note Taker Checklist Form -*MSRI*

Name : Rob Stapleton

E-mail Address/ Phone #: jrstaple@ncsu.edu

Talk Title and Workshop assigned to:
Interactive Parallel Computation in Support of Research in Algebra, Geometry, and Number Theory

Lecturer (Full name): Henry Cohn

Date & Time of Event: 1:30 p.m. Jan 29, 2007

Check List:

(✓) Introduce yourself to the lecturer prior to lecture. Tell them that you will be the note taker, and that you will need to make copies of their own notes, if any.

( ) Obtain all presentation materials from lecturer (i.e. Power Point files, etc). This can be done either before the lecture is to begin or after the lecture; please make arrangements with the lecturer as to when you can do this.
working on it!

(✓) Take down all notes from media provided (blackboard, overhead, etc.)

( ) Gather all other lecture materials (i.e. Handouts, etc.)

(✓) Scan all materials (notes) on PDF scanner in 2nd floor lab (assistance can be provided by Computing Staff) – Scan this sheet first, then materials. In the subject heading, enter the name of the speaker and date of their talk.

Please do NOT use **pencil** or colored pens other than black when taking notes as the scanner has a difficult time scanning pencil and other colors.

Please fill in the following after the lecture is done:

1. List 6-12 lecture keywords: parallel, interactive, wish list, GIMPS, SETI@Home, Folding@home, ~~[illegible]~~ course grain, server, task farming, BOINC

2. Please summarize the lecture in 5 or less sentences.
We wish to get a general idea of what this community wants and plan a way to make tools such as BOINC available to the "casual user."

Once the materials on check list above are gathered, please scan ALL materials and send to the Computing Department. Return this form to Larry Patague, Head of Computing (rm 214)

1:45 p.m.
Parallel Computation Tools for Research: A Wish List
Henry Cohn

No technical content. Just a springboard for discussions.
Cohn is a user, not an expert in parallel computation.

Large computations are increasingly important in mathematics. Computer-assistend proofs. GIMPS

Often embarassing parallelizable. Usually carried out by ad hoc means.
Lots of things that non-experts would like to do b/c barriers are just slightly too high.
We shouldn't build more powerful and sophisticated tools (all the time), but also focus on easy-to-use-for-everyone tools.

Three Scenarios:
1) GIMPS model of recruiting many volunteers
2) Use among collaborators simply for organizing a large-scale computation. (more concerned about)
3) Use on multicore computers.

It's a real pain to get parallel computing up and running.

BOINC is a nice system set up to manage many computers, but not good for the casual user.

If you limit yourself to simple programming in a common file system amongst computers to pass information back and forth, you can't just scale past that file system.

What we really need is a flexible system for automating distribution over many computers (only when trivially parallelizable)
The particular problems in mind are not low-latency, fine-grain problems.
Many assumptions are being made behind the curtain here. Assuming that these problems will be similar to most of the
problems that other general users will encounter.

Server must be trivial to set up. No communication between clients.
Clients can download and return answer or fail.
Problem times out if client times out. Authentication can be required for clients.

There are some open Darwin-based things that are already doing this, it seems.
What this wishlist is concerned with is generalizing this to run on a heterogeneous system and a few other things.
Changing this to an easy-to-use system is one of the biggest goals.

Authentication problem with TCP/IP.

Usage: Random volutneer arrays can cause a lot more problems than they're worth for the common user--mostly arising from
issues of safety and authentication. We don't have to create a massive framework to achieve most of our goals.

Issues for clients: Malicious code, stability, resource hogging, suspendability.
Critical for any volunteer or corporate users.

Sees only one realistic option: Sacrifice some overhead to run these calculations on a virtual machine.
Users don't have to worry (as long as there is a reasonable implementation of a virtual machine) about security issues.
Can be suspended at will.
This is "virtual machine" in the sense that we have a self-contained running environment that never escapes to the computer.
This is no sinecure, but can reduce worries.

The idea is that people are always going to want to exceed the resources they have available.

Virtual machines are not always slower. SAGE runs more quickly in OS X or Windows with a Linux VM.

Spammers may want to use this for malicious spam-oriented purposes. Feels like there's not too big of a risk.

One vision for the future: Log into a server, get a list of jobs running on the server, select which job(s) to join.

Server: DDoS attacks seem to be primary risk. Do what we always do.

Tampering with results: Mischief (user shows up with bogus answers)
Sabotage (users flood bogus answers to corrupt research calculations)
Fraud (exaggeration)
--It is unclear exactly how much we can deal with these problems.

-Consistency checks. Farm out problem to several clients to check answers. Ask about CPU time, resource use, etc.

Risks of tampering depend on anonymity.
Use of anonymous volunteers almost always invites certain risks due to the anonymity.

Should we trust code blindly? It -should- be okay, but errors can crop up even in "good" code.

Volunteers and the program(s) should always be credited. If anyone made a critical contribution, they should be individually acknowledged as well. The finer details of this should be left to community consensus.

Should there be a cutoff for acknowledgement? If you provide 10% of the computing power, should you be credited

individually? It could turn off individual users, since those 10%'s will be provided by clients with supercomputers.

How do we measure the "computing power?"

Maybe the clients can make demands from the server; make the server enter into an agreement where if the client

provides enough, the client will be credited.

A community consensus would be nice, but these other options are flexible.

Within 10 years, the principal application will be massively multicore machines. We don't even know how to build a good computer algebra system for one of these machines.

Speculative execution? If the user starts typing in a function, will some of the cores start plotting the function? Start calculating other things about the function? Then, if the user wants any of that, it will already be computed.