GridCrafty

Jeroen Laros

29th March, 2002

Contents

1 Introduction

GridCrafty is a shell script which parallelises a chess engine named Crafty. Crafty is the best free chess engine available. See Bob Hyatt's site: ftp://ftp.cis.uab.edu/pub/hyatt/ for the latest version. It shouldn't be hard to use another chess engine like Gnuchess, because the script uses Xboard compatible commands. Xboard (Winboard) is a common chess interface so all Xboard compatible engines should work with no or few changes. I assume some rudimentary Globus, UN*X and networking knowledge is known to the reader.

2 Implementation

GridCrafty is generally divided in two modules; a server and a client. The server takes as input a chess board and information about who is to move (black or white) in FEN format (a simple readable way for chess positions). It then analyses the board and comes up with all possible moves. These are then given to the clients who grade the positions. After all moves have been analysed, the server selects the best move.

2.1 Version 1

The first version of GridCrafty used Globus for resource discovery and task deployment. The tasks could only be used on a local system, because NFS was used for communication between client and server.

2.2 Version 2

In the second version ssh in combination with RSA authentication was used for file transfer, so that remote, and therefore more, machines could be used. This would cause some problems later on, but because Globus was still used, the algorithm was still to slow to see the problem.

2.2.1 Version 2.1

This is actually the last version tampered with. It's included because they made me, granted, it was part of the assignment. In this version GASS is used for communication. The major change in the interface is that before you run your first session, you'll have to run the init script. And when you're done, you'll have to run the shutdown script (not between two jobs!).

2.3 Version 3

The last versions use my own algorithms for resource discovery and task deployment. Here is where I ran into trouble. Not all file transfers seemed to succeed. After some hard thought I concluded it had to do with the maximum number of connections of the machine my server was running on. Because the algorithm

was a lot faster now, the server couldn't handle all solutions coming back at the same time. A little wrapper for scp (secure copy) using an exponential back-off policy (inspired by the TCP protocol) solved the problem.

2.4 Version 4

This version is almost identical to version 3 except it is a lot nicer (it doesn't grab a host and start, but checks if the host is free).

3 Results

The first results were rather disappointing, but as the version number increased, the results became better (except for version 2.1).

3.1 Version 1

When testing on a fairly large problem (22 moves) it took about four minutes to complete, which is not so bad, considering each client at least needs 55 sec to complete (if it is a move worthy of analysis). This roughly translates to a speedup of 500%, but compared to the theoretical speedup of 2200% it's rather low. Further analysis showed that the bottleneck was on the server. This could be shown by printing out the starting time of the calculation minus the deployment time (startup overhead). Note that ntp must be installed for this to work. The variation in this startup overhead was huge. It varied between 11 and 173 sec. So a speedup could be expected when using multiple Globus job-managers.

3.2 Version 2

So using more servers is what I did, by deploying tasks to different servers in a round-robin fashion. An additional problem was communication. I first investigated the Globus-toolkit options, which were GASS, globus_rcp and openssh with ssl support. GASS didn't seem to be secure on first glance, globus_rcp wasn't installed (or removed from the distribution) and openssh didn't work. Eventually I chose for scp (I gathered I've bothered the system administrators enough.) and took into account that when a job was run locally (and by locally I mean on the same file server) no file transfers were necessary. This approach gave some better results. On the same problem described above the grid took approximately 2:15 min to finish. I couldn't do the extensive timing analysis anymore because ntp wasn't installed on all servers and nodes, but from the nodes who ran ntp I concluded that the startup overhead now varied between 7 and 60 sec. The total speedup was now about 900%.

3.2.1 Version 2.1

Setting up a local GASS server (which is secure after all) didn't work because the clients don't recognise globus commands, which, I imagine, is normally the case. If it were, no further changed would be necessary. I had no such luck. Remote GASS servers seemed the only other option. Note that this isn't the preferred approach, because normally most hosts will be in different file spaces. Several scripts were needed to set up and destroy remote servers. Because the file servers I was testing on didn't have ntp configured, a large timeout had to be inserted between initialising the local proxy and the remote proxies (Globus gatekeeper doesn't accept certificates from the future). Fortunately this is a once-per-session command. Another problem that had to be solved is the cleanup afterwards. In other versions the client does this. In this version a cleaning job is scheduled after all jobs have finished. This version is slower than version 2 (the file transfer anyway) because remote jobs have to be finished first, in other versions results come in faster than the jobs. The total speedup is about 710% (and I didn't even take the scheduling of the cleaning jobs into account). Note that if the preferred approach was possible, I guess no real difference between version 2 would be measurable (even though a proxy must be set up at the client side). At this point I also discovered that some of the nodes I was working on got reinstalled. Normally that's a good thing, but in this case the standard C library (amongst others) was gone, which prevented the engine to execute. Statically linking the Crafty executable solved this nuisance. One last note on the code: I didn't generalise the setup, shutdown and cleaning of temporary files because this isn't a normal situation.

3.3 Version 3

By bypassing Globus entirely, even more speedup could be expected. To do this, some form of resource discovery was needed. This was done by simply connecting to all known nodes, and querying the load. If the load is less then 10% (assuming the query itself increases the load), the host is probably free. With this, rather aggressive, approach a speedup of about 1750% was measured. The startup overhead was reduced to 1 sec. A disadvantage to this approach is that this is static. A better approach is to build in a test in the client so that is fails (and gets rescheduled) if the load on the host is too high. At this time this isn't implemented because the nodes always seem to be free (it is also a bit beyond the scope of the assignment). Sometimes after an hour or so of testing, some jobs begin to slow down due to reserved nodes. A rerun of the resource discovery script always solved the problem so far. While debugging this version I also ran into the sshd overload on the server. The solution to this problem (mentioned above) was also incorporated in version 2. Another method to reduce sshd overloads would be to launch exploders on the remote file servers, like I did in version 2.

3.4 Version 4

I went along and did it anyway.. In this version the client exits if the load is too high just before the intensive calculations start. It doesn't seem to have much effect, occasionally a job gets rescheduled, but when the average load on the grid increases I suspect this part of the code will be reached more often. With a similar adjustment a kind of heartbeat can be implemented, but the reliability of the grid I'm currently testing on is fine.

4 Conclusion

Because I've only tested the scripting abilities of Globus, no real conclusion about the performance of Globus can be given. However, in my case, a problem-specific solution is better than the Globus solution. For some reason the scheduling of tasks takes up a lot of time. GASS is fast enough, although the initialisations are bothersome. I'm impressed by the speed of ssh, which, by the way, can still be improved by using faster encryption algorithms, blowfish for example. A better improvement would be backgrounding the deployment. If that works, the maximum speedup would nearly be reached, however, it doesn't. For some reason authentication must be done first, then the ssh session is allowed to fork. I've also got the faint impression Globus isn't intended for high-end clusters like the one I'm testing on, but I may be wrong. Fortunately I'm not the only one testing Globus and I'm interested in findings of others.

5 Thanks

I'd like to thank David Groep for the countless times he gave me a new Globus certificate and helping me out with GASS. I'd also like to thank Timo Krul for telling me the ssh RSA secret.