Multinode Sequencing within NetSolve

Jakob Østergaard
(jakob@unthought.net)

Innovative Computing Laboratory
University of Tennessee, Knoxville
and Oak Ridge National Laboratory

Department of Mathematical Modelling
Technical University of Denmark

February 3, 2001

Several “Grid Computing Systems” exist and are being developed today. These are distributed computing systems, that enable users to distribute large scientific computations over a diverse network of computing resources, called “The Grid”.
This report describes the efforts made to expand the functionality of one of these systems, to allow the distribution of sequences of computational problems to multiple nodes on The Grid.

The author spent 3.5 months as a visitor at the Innovative Computing Laboratory at the University of Tennessee, Knoxville. This time was spent working with the NetSolve group on various aspects of NetSolve, with the main focus on introducing multi-node sequencing into the NetSolve system.

The author would like to thank Distinguished Professor Jack Dongarra, and Professor Per Christian Hansen for making the visit possible. Further thanks go to the people of the NetSolve group, especially Michelle Miller and Dorian C. Arnold as well as the other visitors and ICL staff for making the stay enjoyable and rewarding.

This report and the complete source-code for the modified version of NetSolve is available from the author's web-site (see [5]).

1 Background

The NetSolve grid computing environment (see [2]) enables users to solve numerical problems on remote compute servers running the NetSolve system.

1.1 Sequences of problems

The NetSolve system works with "problems" (such as, for example, a matrix multiply, or a linear system solver) – a client program can submit a problem along with it's input data to the NetSolve system, and the system will then return the resulting data to the user after successful computation. The user is not concerned with *where* the computation is performed – it is the job of the NetSolve system to find and utilize available computational power on "The Grid" (the network of computers making up the NetSolve system). See [1] for an introduction to the concept of "The Grid".

Often it is desirable to submit a number of problems – a *sequence* – in which some problems would depend upon results from previous problems in the sequence. Originally, users would have to submit each problem at a time, but this would incur an unnecessary overhead as intermediate results would be passed back over the network from the server to the client (the user program), and once again back to the server as input to the next problem.

To overcome this wasteful way of executing multiple (and individually dependent) problem solvers, a *sequencing* subsystem was introduced into NetSolve. This basically enables the user to tell the NetSolve system that she wishes to solve a sequence of problems, instead of simply giving the NetSolve system one problem at a time. NetSolve can then consider dependencies of input and output data from the problems in the sequence, and execute these on a single server, without the need for transferring un-needed (and unwanted) intermediate results back to the client.

While the sequencing approach is superior to the simplistic single-problem submission, the work done in order to make sequencing possible opens the doors to another way of gaining speed-up in computations involving long sequences of problems.

1.2 Multinode sequencing

In the “official” NetSolve release, a sequence can only execute on *one* NetSolve server. If multiple clients each submit sequences, those will of course execute concurrently on each their server, but once a sequence is submitted to a server *every* problem in the sequence will be executed on exactly that server.

Sequences can easily contain independent problems and these can execute in parallel on that one server, giving a performance boost if the server has multiple execution units (processors). On a single-processor server, such parallelism in the sequence cannot beneficially be exploited.

However, in a grid environment where processing power is immense due to the abundant availability of compute servers, it seems highly likely that the sequencing approach of computation could benefit significantly from the ability to actually execute independent problems in the sequence on distinct servers. This, however, is not possible in the current sequencing implementation in NetSolve.

This report describes an effort made to provide a prototype-implementation of such “multi-node sequencing” capabilities in the NetSolve system.

2 Goals and considerations

For various reasons, the author decided to only implement a prototype of multinode-sequencing into NetSolve, rather than providing a full-blown production quality extension to the system.

2.1 Immediate goals

The immediate goals for the multinode-sequencing work can be outlined as follows:

- There should be no change in the user-visible API that exists for single-node sequencing today
- Multi-node sequencing should be used automatically by the NetSolve system for sequences that can benefit from execution on multiple nodes
- Use of standard technologies wherever possible

The first one of these goals should be easily achievable. One of the virtues of the NetSolve system is, that the user does not need to concern herself with which servers to use. Of course, this should still be true when a sequence is distributed over multiple nodes in the network. The infrastructure for this abstraction has been in place for a long time already, so there should be little or no extra work required in order to fulfill this.

The second goal is almost a side-effect of the first one. The existing data dependency analysis already used in sequencing, will be used to determine – at run-time – whether a problem can safely be solved on another server.

A separate goal of the implementation itself will be some amount of modularity in the code. It must be possible to experiment further with the decision algorithms working behind the scene of the sequencing API, as this will obviously be an area in which much work could be done – Especially if the NetSolve system later on will be equipped with topological information about the network it is running on, and more verbose resource availability information.

The third goal seems obvious. It is mentioned here because it significantly affects the way a part of this work has been carried out. As will be explained later, the *Internet Backplane Protocol* (IBP) is used for server-to-server communication in this work. IBP is an ICL project like NetSolve.

2.2 Inherited problems

As the work described here is being done on the existing NetSolve system that has been under development for years, there is a large heritage of functionality and – unfortunately – problems.

The NetSolve code-base has grown rapidly over the years, and today NetSolve consists of more than 100.000 lines of `C` code. The system is fairly complex, as a large number of different processes are running at different places in the network, and each of those will be communicating with a number of the others. This, combined with a lack of documentation of the inner workings of the system, makes it a challenging task to extend functionality deep within the NetSolve core.

However, it has indeed been possible to extend the NetSolve system to do multinode sequencing. The difficulties met in the process is the reason behind not providing a production-quality multinode-sequencing implementation. It was simply too much work to properly re-structure a large part of the NetSolve core, in order to fit in the new functionality in a way that would be easily maintainable and obviously correct. The implementation that was made is neither of those – it is however good enough to show whether multinode sequencing is feasible, and it will point us toward new problems and bottlenecks in the chosen design.

3 System overview

As can be seen in Figure 3, quite a few entities are involved when a client submits a problem to the NetSolve system. From the figure we can count six different network protocols and five inter-process communication protocols – a total of eleven independent protocols for communication within the system. There are even more protocols, as Figure 3 only shows a snapshot of the system when running a problem solver.

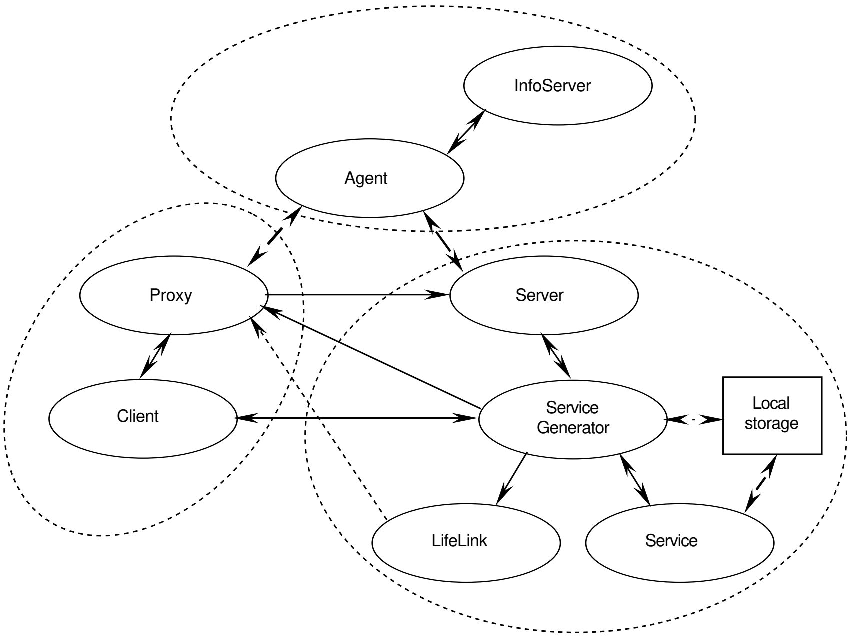

Figure 3.1: Overview of the "original" NetSolve system during problem solving – each bubble represents a running process in the system. The dotted lines represent physical computers on the network and shows which process is running on which machine.

While Figure 3 does not give a complete picture of all entities in the system, it shows all entities that have been modified directly as a consequence of introducing multinode sequencing.

3.1 The breakdown

The purpose and function of the different parties is explained below:

Client This is simply the client program that uses NetSolve. It is a user-supplied program that makes calls to NetSolve using the NetSolve `C` or `Fortran` interfaces.

Proxy This process is spawned by the client program as it initializes the NetSolve library. The purpose of the proxy is to allow a NetSolve client to use other computational back-ends than NetSolve, such as Globus, Condor and Legion. The proxy is an abstraction layer, so to speak.

Agent The agent is the central book-keeping authority in the NetSolve system. Currently, only one agent can exist in a NetSolve system, no matter how many servers and clients is set up. The agent keeps track of resource (load) information from the servers, and it maintains a list of which servers are on-line and which problems each server can solve. The agent is a directory of meta-data, and is the sole source of information for other parts of the system when job distribution or scheduling is carried out.

InfoServer The information server is a networked log database. The agent will log all information regarding job distribution and completion, server registration, etc. Monitoring systems can then access this information using a simple query language over a standard TCP connection. More information is available in [3].

Server The server process can be running on a large number of machines on "The Grid". It is intended to be run on a wide array of heterogeneous computer systems on a wide area network (this constellation is referred to as "The Grid" – see [1]). The main responsibility of this process is to initially register the list of problems it can solve to the agent, and thereafter receive and service job execution request.

Service Generator This process is forked off by the server process. It's responsibility is to receive all input data for the problems at hand and write these to persistent storage. It will then start up one "LifeLink" process, and a service process for each problem to solve. This process also notifies the proxy of job completion and sends back results from persistent storage to the client.

LifeLink The LifeLink is a process that maintains a TCP connection to it's parent process (in this case the service generator). If the connection is closed, which usually means that the parent process died unexpectedly, the LifeLink will send back status information to a previously defined remote process. In this case, the LifeLink will send back an error report to the proxy if the service generator dies – the LifeLink thus serves as band-aid for programming errors in the service generator.

Service Each subset of problems (eg. LAPACK routines, BLAS routines, etc.) has their own service program. The service generator will spawn the appropriate service program once the input data needed to solve the problem has been written to persistent storage (either sent from the client over the network to the service generator, or written out as computational results by a previous service process). Once a problem is solved, the output data (the results) are written back to persistent storage.

4 Required changes

In order to introduce multi-node sequencing, a number of changes were required deep within the NetSolve core. Some were absolutely necessary, others are optimizations that can significantly improve the "throughput" of problems both with and without multi-node sequencing.

4.1 IBP integration

Originally when solving a sequence, the service generator would receive input data for the problems from the client and write this to local disk storage. The service processes could then read the input data from disk, solve the given problem, and write back results. After the whole sequence was complete, the wanted results would be read from local disk by the service generator, and sent back to the client.

This had to be changed in order to accommodate multi-node sequencing, since multiple servers would need (eventually concurrent) access to the data. We could rely on having the data on a networked file-system, but that is a poor solution because of the extra administrative burden, and because of the inflexibility (a user experimenting with NetSolve would have to involve the systems administrator in order to get networked file-systems configured).

The solution chosen is to use the *Internet Backplane Protocol*, IBP, as the way to get this networked persistent storage. Since IBP is an ICL project like NetSolve, the author had good opportunity to discuss IBP details with the IBP group during the development of multi-node sequencing.

As seen in Figure 4.1, IBP is simply "plugged in" as persistent storage instead of the local disk storage used previously. However, the change was by no means simple. The internal API for object (input and output data) communications was not intended to be expanded to utilize another low-level API such as the IBP API. The existing framework for communicating input and output data cannot be extended elegantly to use a communications channel that is not described with a file-descriptor (such as a socket or a disk-file) and works with standard `read` and `write` system calls. This is not a critique of the way the NetSolve communications layer is written, as the restrictions are perfectly reasonable. It does however explain why the implementation of multi-node sequencing is not suitable for direct inclusion into the NetSolve project.

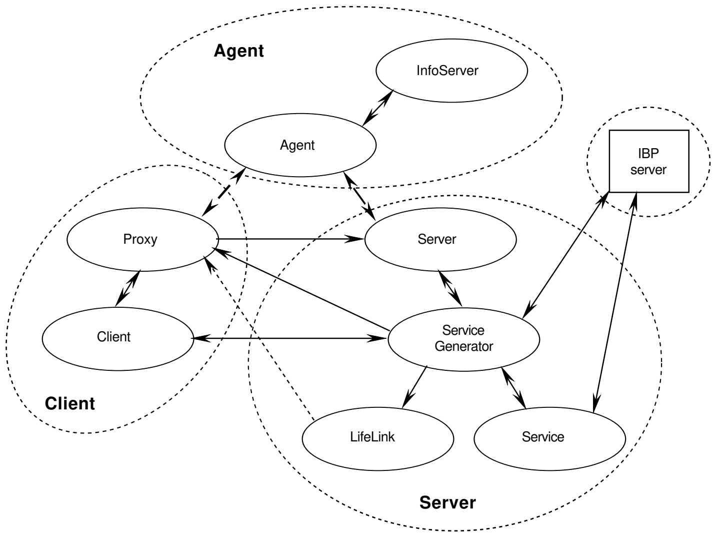

Figure 4.1: The IBP server replaces the disk-files as persistent storage for problem input and output data, thus allowing for multiple servers to access the data

The communications layer was partly duplicated, and the duplicate then refitted to use the IBP API instead of the POSIX API as used by the original communications layer. This means, that thousands of lines of code was duplicated. The alternative to this duplication of code would be to properly abstract the communications layer, and then fit it on top of the IBP API or POSIX API as needed in the different parts of the NetSolve code. This was deemed to be too much work to complete in the fairly short period in which the multi-node sequencing work was completed.

Looking back though, properly abstracting the communications layer would have been the right thing to do. This is a change that should be done before the multi-node sequencing is included in NetSolve.

One more problem with the current state of the IBP integration is, that we unconditionally use IBP for storage of input/output data. It may not be needed to use IBP at all, if the sequence is kept locally. Using IBP is likely to be slower than using local disk, because the operating system can buffer writes and cache reads (thus, for input/output objects that are significantly smaller than the available physical memory in the machine, read and write speed can approach the memory-bus bandwidth, or some large fraction thereof). IBP itself is also relatively slow today – however, IBP is still a work in progress, and if the IBP server and libraries can be optimized to satisfaction, the multi-node sequencing will also benefit from that.

4.2 Completion polling → notification

NetSolve used a polling strategy between the client and the proxy in order to find out when a submitted job was completed. The client would poll the proxy every 3 seconds to see if the service generator had notified the proxy of job completion.

This is a wasteful and inefficient way of implementing a simple inter-process communication. The polling scheme was changed to a completion notification scheme. When the client waits for job completion it will now simply sleep exactly until the proxy tells the client that the job is completed (or has failed).

Furthermore, when the service-generator is working it's way down thru a large sequence, it will also send back partial-completion events to the proxy (which will be relayed from there to the client). This would enable the client to show progress reports to the user during processing of large sequences, etc. However, none such functionality is implemented in the NetSolve client library yet.

4.3 Global job-id tracking

Previously, a submitted job would be assigned an identification number (ID) by the proxy. Naturally, the proxy would not be aware of which numbers were assigned by other proxies (spawned by other clients perhaps on other machines), so the ID is only unique in the proxy that assigned it. This ID scheme has been preserved because a large part of the existing code depended on these IDs.

In order for the agent to log meaningful information about sequence job submission and completion to the info-server, it would be necessary to ensure that every submitted job had an identification number that was unique in the entire NetSolve system at any given time. This number is called the job-id.

The agent is the responsible party in assuring that a job-id is truly unique (at a given time – numbers are recycled eventually). The proxy will request job-ids from the agent and assign these to the jobs in a sequence before sending job requests to a server. As every proxy will do this, it is possible to guarantee that the job-ids are unique.

4.4 Server request addition

The simplest way to split out a sequence over multiple servers, was deemed to be by letting one server receive the full sequence request and letting that server request that others solve particular problems in that sequence.

This is a very dynamic approach, as the decision about who will execute which job, is made immediately before the decision is acted upon. An alternative approach would be to assign jobs to servers initially, eventually letting the agent handle this, and then strictly follow these initial assignments. However, a running NetSolve system is by nature dynamic. No part of the system (not even the agent) can know (or predict) if or when a user should decide to submit yet another problem to the system, or of what nature that problem might be. Minimizing the number of long-term decisions seems to be the right way to go.

A request was added to the server, so that it can not only accept job submissions from clients (proxies actually), but also from other servers. This way, a server that is assigned a long sequence can – dynamically as it deems necessary or feasible – request that other servers solve parts of the sequence.

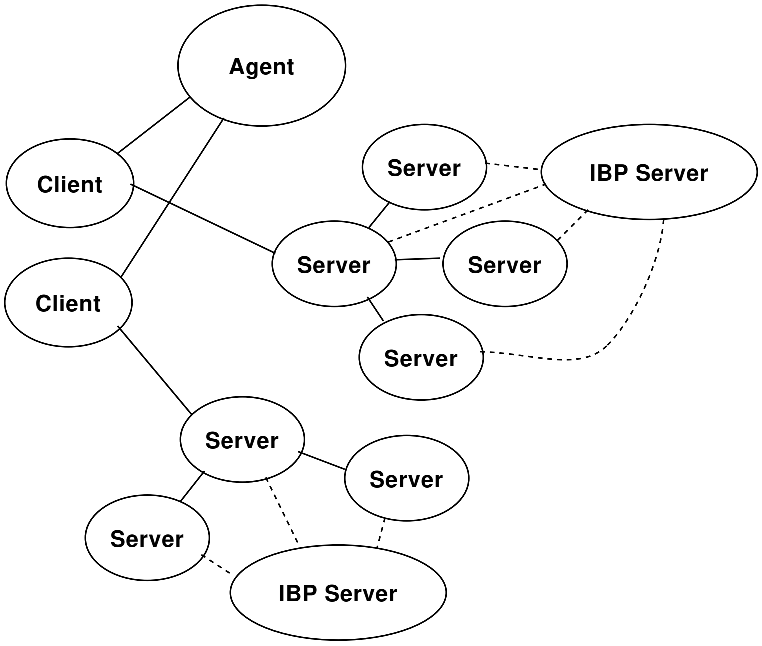

Figure 4.2: Draft showing two clients each submitting a sequence to the NetSolve system. The sequences are being solved on two different groups of servers, each with their own local IBP server. Each group has a "master" server and multiple "slaves"

In Figure 4.4 a NetSolve system solving a sequence from each of two clients can be seen. Details about the IBP servers is given in Section 4.1. Notice how the client contacts exactly one server which then will contact other (slave) servers. Also notice that all servers working on a sequence share the same IBP server. This will be explained further in Section 4.5.

4.5 Grouping

It was decided to use IBP as the underlying mechanism for server-to-server communications during sequence solving. This implies, that all input and output data to/from a problem solver (service process) will travel over the network from/to an IBP server.

In order to minimize the performance impact from this extra network use, we ensure that the IBP server used is placed topologically (network-wise) close to the servers involved in solving the sequence.

Since it is very difficult (or impossible – depending on one's requirements for precision) to measure such topological properties with good precision, a new option was added to the server configuration file. In the server configuration, the administrator will now specify which IBP server (if any) is local to that server. When the server registers with the agent, it will also let the agent know which IBP server is close.

When scheduling a sequence, the proxy requests a lot of information from the agent – among that, the lists of problems each server can solve, and which IBP server is local to each NetSolve server. Based on this information, the proxy will make *groupings* of servers, where each group share the same IBP server. It will then find a group where, for each problem in the sequence, there is at least one server that can solve the problem.

This logic should let the administrators of NetSolve servers minimize the network utilization by configuring IBP servers on each segment (or "subnet" or other topological entity on the network), and making sure the servers on that part of the network use that particular IBP server. A submitted sequence will then automatically be assigned to a server in a group of servers that all share the same topologically close IBP server, and where the servers combined has problem solvers for each problem in the sequence.

Which group is chosen, if several groups are valid candidates (eg. several groups can solve the sequence), is an area where much more experimentation can be done. Currently, the group with the most machines is chosen. This will be sub-optimal in a number of situations of course, one should also consider the availability of resources such as CPU power and memory. Even more metrics could be taken into account, and more sophisticated groupings could be done. For example, if no group can solve the entire sequence, it may be possible to have two groups co-operate – the IBP API even offers convenient ways of requesting that data be moved from one IBP server (local to one group) to another IBP server (local to another group).

5 Results

As touched upon earlier, there is a number of known weaknesses in the way multi-node sequencing is implemented in the work described here. However, the weaknesses are easily identified, and some can be resolved within reasonable effort. The results presented here shows the benefits of multi-node sequencing, and clearly points out where optimization efforts should be focused if multi-node sequencing is implemented in a final form, using techniques and design similar to that presented here.

5.1 The test environment

All tests were run on the `torc` cluster, the *Tennessee / Oak Ridge Cluster*. The cluster consists of eight dual-CPU 550MHz Intel Pentium-III machines with 512 MB of RAM each. The interconnect is switched Fast Ethernet.

The cluster was used during night, so that the load on the network and file-servers would be minimal. Also, it was ensured that the nodes used for these tests were idle at the time of testing.

5.2 Baseline overhead

Tests were made to clarify to what extent the changes described in Chapter 4 has on performance, when solving a sequence that *cannot* take advantage of the multi-node sequencing. Such a sequence will simply be a list of problems where each problem depends on the result from the former problem. With such a dependency, NetSolve cannot execute two problems concurrently (since one problem would miss input data), and thus cannot take advantage of running jobs on multiple servers.

Figure 5.1 shows the elapsed run-time of solving the same problem (two matrix multiplications) with and without IBP integrated into NetSolve. The matrix dimension varies from 50 to 200. The amount of data transferred varies as $\mathcal{O}(n^2)$ and the computational work done varies as $\mathcal{O}(n^3)$.

It is seen, that in this particular case, using IBP is actually to be preferred over using local disk storage. However, the largest difference in run-time is on the order of 8%.

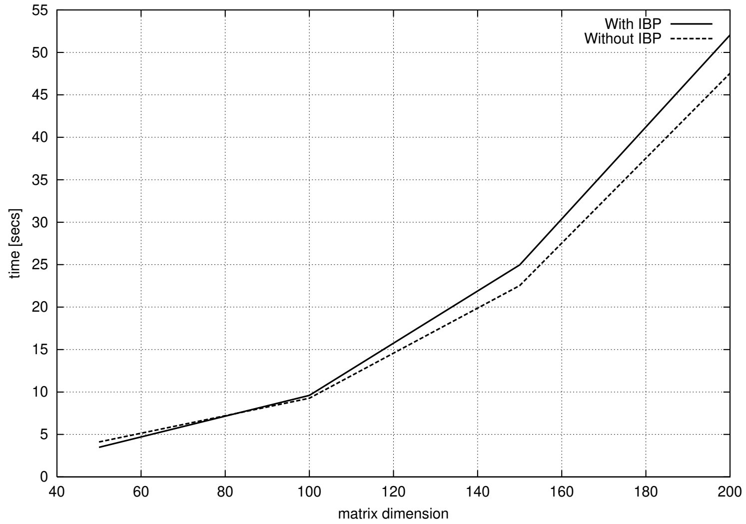

Figure 5.1: Sequential problem – elapsed computation time with and without IBP integration

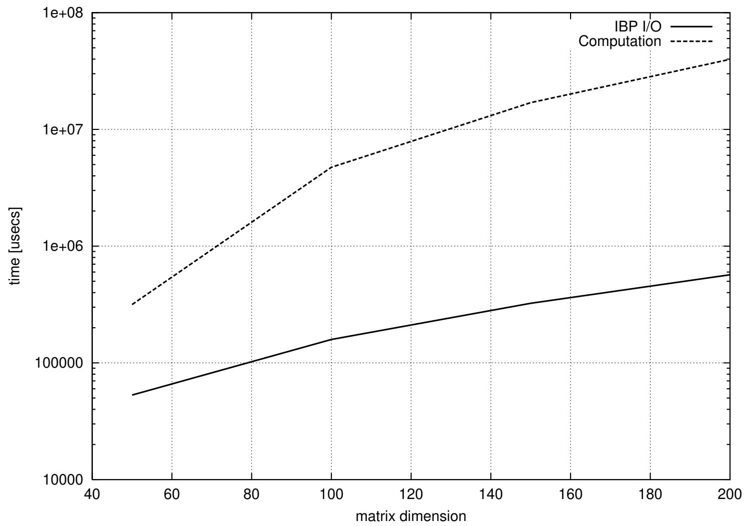

Figure 5.2: Time spent doing IBP I/O versus time spent in computation

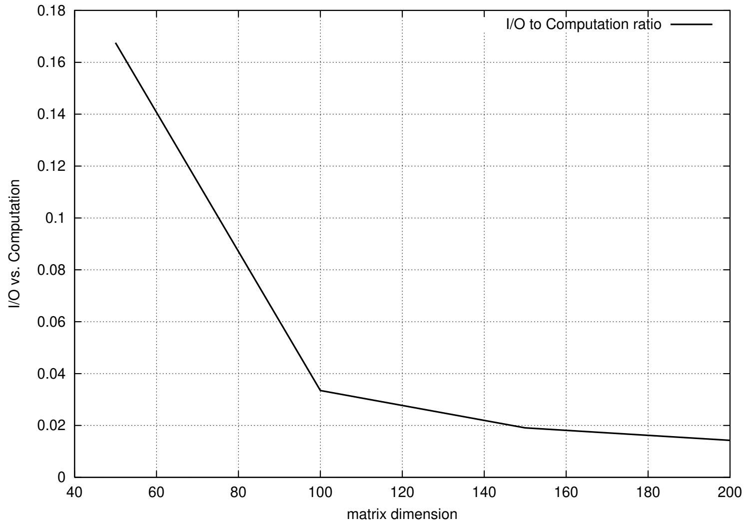

Figure 5.3: The ratio of IBP I/O time over computational time spent – corresponding to Figure 5.2

Figure 5.2 shows the time spent doing IBP I/O and the time spent in the server doing actual computations, as the problem size changes. A logarithmic second axis was used in order to visualize both lines on the same graph, as the computational time rises n times faster than the I/O times.

In Figure 5.3 we see how the time spent doing IBP I/O relates to the time spent doing actual computations. We can see, that as the amount of FLOPS needed grows compared to the amount of data needed, the actual communications overhead shrinks as well.

These graphs show no surprise, but it confirms that the use if IBP within NetSolve provides a method for transferring data to computational nodes, that behaves much like one could expect. More computation on less data yield less communications overhead, as usual. Furthermore they show that the actual overhead from IBP is relatively low – and Figure 5.1 confirms that the over head is also low when compared to the existing mechanism in NetSolve for communicating data.

5.3 Multi-node speed-up

Tests were performed to see how well the improved sequencing system will scale to multiple machines, when put to work on a problem that will actually scale to the number of available servers. The sequence chosen for this test is actually just N independent copies of the sequence from the previous test, concatenated into one long sequence. Such a sequence will scale perfectly to N servers, if we do not consider the transportation of input and output data.

In Figure 5.4 we see how the multi-node sequencing speed-up behaves for a matrix dimension of 50. We submit a sequence that contains a number of sub-sequences that are mutually independent and equal to the sequence treated in Section 5.2. If the system was scaling perfectly, the elapsed time should remain constant as long as the number of sub-sequences is less than or equal to the number of servers.

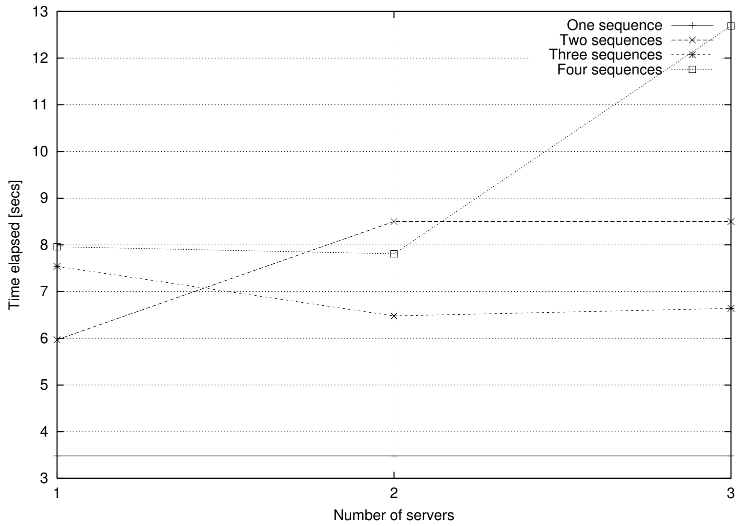

Figure 5.4: Elapsed time for various number of concurrent sub-sequences on various numbers of servers. Matrix dimension is 50

Surprisingly, Figure 5.4 shows *very* poor scalability. Actually, with four sequences the elapsed time *increases significantly* as we move from two to three servers ! Before concluding anything, let's look at the same test with a larger matrix.

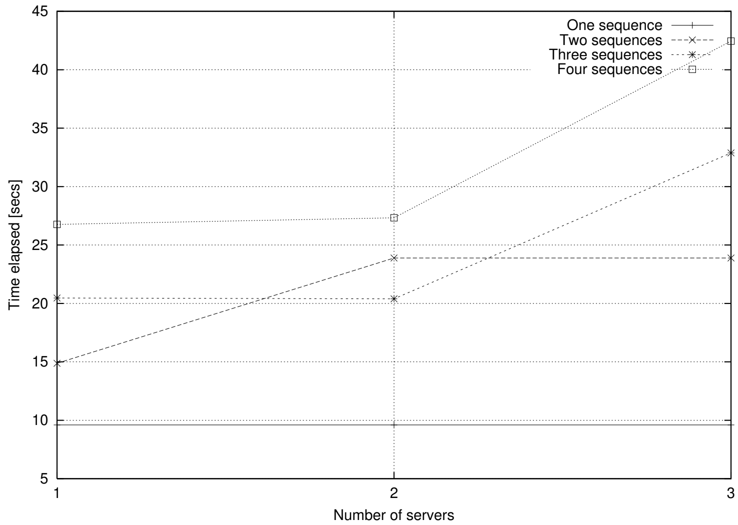

Figure 5.5: Elapsed time for various number of concurrent sub-sequences on various numbers of servers. Matrix dimension is 100

As seen in Figure 5.5, the scaling is even worse for larger matrix dimensions. Both the three and four concurrent sub-sequence cases *significantly increase* their run-time when we move from two to three servers ! This calls for further investigation.

Let us look at how much time we spend on communication, in the various runs that show problems. It seems likely that we may be hitting a scalability problem in the network, a problem with the way that IBP was integrated into NetSolve, or in IBP itself.

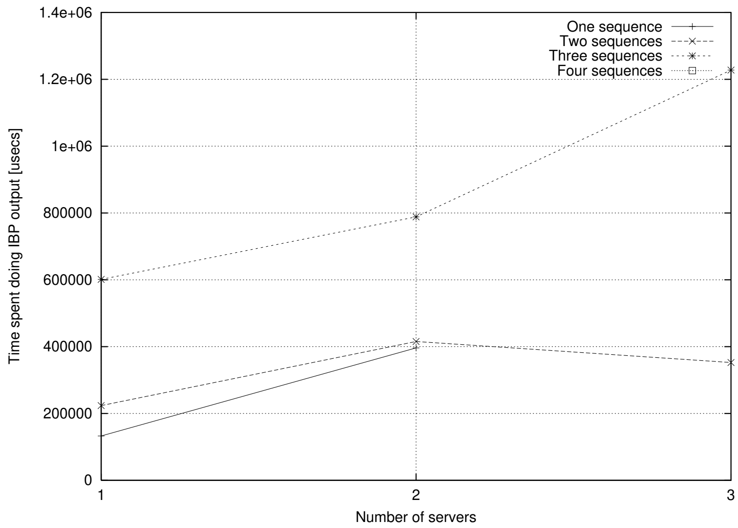

Figure 5.6: Time spent in `IBP_store()` calls, running the multi-node test with matrix dimension 50

In Figure 5.6 we measure the time we spend writing data back to IBP storage, with the same test as shown in Figure 5.4. The matrix dimension is 50, and Figure 5.4 showed us that we had the most severe scaling problem in the case with four sequences going from two to three servers. We see from Figure 5.6 that this problem can indeed be caused by our use of IBP, as the time we spend in the `IBP_store()` call for writing results back to IBP storage increases significantly in exactly that case.

But why the sudden increase ? Let us look at the time spent in each series of calls to the IBP store routine. Since NetSolve uses a very large number of calls to this routine, the calls have been timed collectively, so that we only measured the time spent in writing out the collection of results from each completed computation. These measured times are shown in Table 5.1.

We can see in Table 5.1, that storing some results take considerably longer time than storing others. This is odd, since all results are of the same size ! It seems, that we have encountered a problem with the scalability of the IBP server, as Figure 5.6 clearly shows that the IBP communication times worsen as more servers are added. If IBP scaled, the total communication time should be equal no matter the number of servers, since the number of calls to IBP and the amount of data transferred is exactly the same.

To confirm that it is indeed IBP scalability that is giving us a huge problem, we look at the time we spend in `IBP_store()` in the case with a matrix dimension

Store #	Node 0	Node 1	Node 2
0	93420 μsec	**213691** μsec	**124978** μsec
1	63721 μsec		**140313** μsec
2			69974 μsec
3			**118747** μsec
4			69975 μsec
5			90893 μsec
6			**124782** μsec
7			59229 μsec
8			58012 μsec

Table 5.1: Time spent in each call to `IBP_store()` in the multi-node test with matrix dimension 50

of 100. This graph is seen in Figure 5.7. Note, that for a fixed number of sub-sequences, the total time spent in `IBP_store()` should ideally be *constant for any number of servers*, since the amount of data is constant.

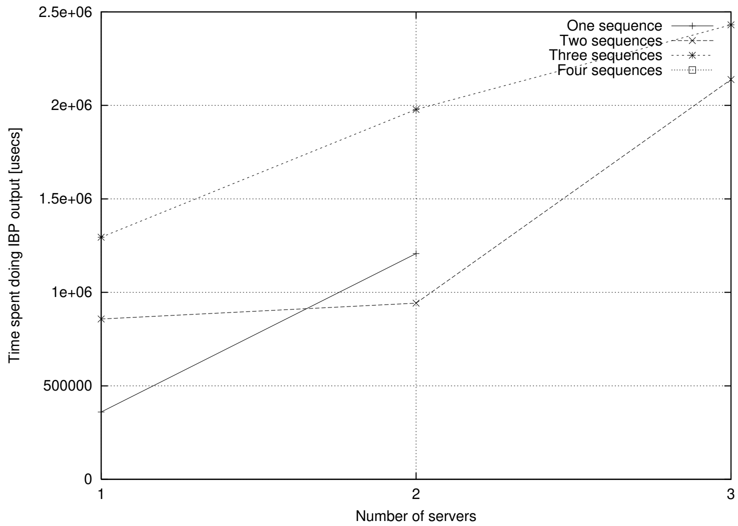

Figure 5.7: Time spent in `IBP_store()` calls, running the multi-node test with matrix dimension 100

5.4 Conclusion on results

It seems clear, from the results obtained in this chapter, that the current implementation of multi-node sequencing simply doesn't scale. While using IBP for input and output data storage still seems like a good idea conceptually, it is clear that the way that IBP was integrated, or IBP itself, must undergo significant changes for multi-node sequencing to useful.

The main problem is caused by scalability problems in the IBP server, as identified in this chapter. It may be, however, that some of this problem can be partially or completely resolved by using IBP in a different way. It is indeed possible, that the scalability problems do not show up if IBP is integrated into NetSolve differently.

It is also entirely possible that the scalability problem is caused primarily by IBP itself, in which case this issue should be resolved with the IBP group.

Due to the relatively short time dedicated to this work, further investigations into the multi-node scalability problem has not been conducted.

6 Improvements

This chapter suggests a number of changes and improvements that could be introduced in the NetSolve and IBP systems in order to make multi-node sequencing more feasible.

6.1 IBP problems

There are three main problems in the IBP system that leads to the discouraging results in Chapter 5.

- All IBP read/store calls are blocking
- Poor implementation (performance wise) of the read/store calls
- Poor scalability to multiple clients

The first problem is an API problem, the second is a pure implementation problem. The second problem is the primary bottleneck in NetSolve's use of IBP right now (partially because of the sub-optimal way in which NetSolve is using IBP – which will be treated in Section 6.2.1.

The scalability problem is of more general nature. It will not be resolved by resolving the implementation problem with the read/store calls mentioned above – but since the IBP implementation itself is outside of the scope of this report, further investigations have not been conducted.

Unfortunately the problems with the IBP integration that can be tracked to IBP itself, were identified too late to get any feedback from the IBP group at ICL.

Other problems discovered during the integration has been discussed with the IBP group, and the author believes this interaction has been mutually beneficial.

6.1.1 IBP API Change

Other systems for sending messages over a network – such as BSD sockets or the *Message Passing Interface* (MPI) – provide a way for the user to *initiate* a transfer, without implicitly awaiting it's completion too.

Non-blocking communication is currently impossible with IBP. NetSolve would not be able to exploit such non-blocking communication constructs the way that IBP integration was done now, but if other changes (described in Section 6.2) were introduced, non-blocking communication could become important. A work-around would be to spawn a worker thread to handle the non-blocking call, but this is an error-prone solution to a problem that other communication APIs handle elegantly with simple non-blocking calls.

6.1.2 IBP Implementation Change

NetSolve uses a lot of IBP read/store calls. It is important that such calls perform well, if we want good performance from the IBP integration in NetSolve.

However, inspection of the IBP implementation reveals, that a new TCP/IP connection is set up *for every single* read/store call made to the IBP server. As setting up a TCP/IP connection is very costly even over a local area network, and since the connection could easily be kept open (even completely transparently to the user with no API change at all), it is hard to find any good arguments why IBP does so.

The IBP implementation should keep open connections once established, or at least cache them with some reasonable limitations for inactivity period or maximum number of cached connections (etc.).

6.2 NetSolve changes

This section describes changes that should be done both within NetSolve itself, and in the way that IBP was integrated into NetSolve. Unfortunately, the resolution of most of these problems will require fundamental changes within NetSolve.

6.2.1 Smarter buffer layout

IBP was integrated into NetSolve by changing the way that NetSolve stores input and output data from problem solvers (so-called "services"). Previously, NetSolve used disk-files on local disks, but this was changed so that NetSolve now uses IBP. The change involved changing a large amount of file read/write calls to IBP read/store calls.

However, the overhead of a file read/write call is very small compared to the overhead of doing an IBP read/store call. By making the simple change we therefore introduced a large overhead in the NetSolve system whenever inputs or results are read or stored.

A way of avoiding this overhead would be to group together a large number of read or write operations into just a few IBP read or store calls. This requires a fundamental change in the way that data are laid out in the NetSolve system.

Each routine in NetSolve should no-longer work on small data structures changing just a few elements and issuing read or store calls on those few elements.

Instead, all the data needed to be communicated should be mapped into one large buffer, which could then be stored (or read) with a single IBP call.

This change would improve performance significantly, but introducing such large shared buffers within the NetSolve system is not something the author sees as an easy change.

6.2.2 Better scheduling support

The scheduling done when distributing jobs from a sequence is extremely simple. We have a simple notion of topology – we know which server share the same IBP server and which do not, which should be an indication of which servers are close and which are not. But this is sufficient only for making rather naive scheduling decisions.

More information about topology and available resources should be available. Furthermore, the scheduling decisions should be handled not on the client side, but by the agent – the only entity in the NetSolve system that can be made to have knowledge of what is actually happening in the *entire* system.

The decision-making of which sequence goes where is currently done by the client, with little support from the agent. The decisions about which jobs in the sequence goes where is done by the server handling the sequence. Obviously this will lead to far-from-optimal scheduling decisions, since two clients will never know about the existence of the other, and one server will never know if another server in it's IBP group may already be busy handling some other sequence.

The information flow and the decision responsibility within NetSolve today is – in the author's opinion – not properly laid out to support good scheduling decisions. Major changes will be required throughout the system in order to centralize knowledge and decision-making.

Conclusion

This report concludes the work done on integrating IBP into NetSolve in order to support multi-node sequencing, the distribution of sequences of jobs to multiple machines in a network.

It is shown, that the "quick and dirty" approach certainly did not pay off with regards to performance. However, a number of problems both within NetSolve, within IBP, and with the way the integration was done has been pointed out. This knowledge will be useful for further work on multi-node sequencing.

It is clear, that multi-node sequencing is a useful feature. Work should certainly be done on NetSolve to properly implement this functionality. This report and the IBP-aware NetSolve system it describes should be useful for avoiding many of the pitfalls in such work.

There has been talk in the NetSolve group to create a "NetSolve 2" system – a version of NetSolve that is significantly changed from the current versions available. If IBP-integration and multi-node sequencing is taken into the early design-stages of this new NetSolve system, it seems possible indeed to create a well-performing multi-node sequencing subsystem. The main problems with the implementation described in this work are trivial in nature, but require fundamental structural changes in NetSolve to avoid.

Jakob Østergaard
`jakob@unthought.net`

Department of Mathematical Modelling
Technical University of Denmark
Lyngby

Bibliography

[1] The Grid: Blueprint for a New Computing Infrastructure
Ian Foster, Carl Kesselman, 1998

[2] NetSolve — The NetSolve Grid Computing Environment
`(http://icl.cs.utk.edu/netsolve/)`

[3] The InfoServer — An Information Server for NetSolve Users
Jakob Østergaard, 2000
`(http://unthought.net/NetSolve/infoserver.ps)`

[4] IBP — The Internet Backplane Protocol
`(http://icl.cs.utk.edu/ibp/)`

[5] The Author's multi-node sequencing resources page
`(http://unthought.net/NetSolve/)`