

02443 Stochastic Simulation Grid Computing

Casper Willestofte Berg (s021692)

Thomas Hammershaimb Mosbech (s011542)

Peter Eriksen (s022018)

June 23, 2006

Informatics og Mathematical Modelling
Technical University of Denmark

Contents

1	Introduction	2
2	System Description	2
2.1	The Real System	2
2.2	The System Model	2
3	Determining Sample Distribution	4
4	Simulator implementation	5
5	Validation	5
6	Verification	6
7	Results	7
7.1	Design of Experiments	7
7.2	Processing Times and the Back-off Function	7
7.3	Slow Computers	7
8	Conclusion	10

1 Introduction

This report describes our work on a stochastic simulation of a chess brain calibration system consisting of several computers for distributing the task processing.

We deal with the following problems:

1. To investigate processing times for k clients and N tasks.
2. To investigate the influence of different back-off functions on the processing time and server/network load.
3. To investigate the effect of having clients with different speeds – especially the case of one slow client.

2 System Description

2.1 The Real System

Our physical system consists of a network of computers, with one central server and several clients. The server distributes batches, where a batch is a queue of tasks to be performed by the clients. All communication is initiated by the clients for various reasons¹. The setup is illustrated in figure 1. Each client asks the server for a task, and if a task is available on the server, it is handed to the client for processing.

In our case each task is a chess position from which two given chess programs will play a full game using a fixed given search depth. When a game is played to the end, the client contacts the server to give it the result. All tasks in a batch should be processed and results returned to the server, before the server starts handing out tasks from the next batch.

If a client asks for a new task, and the server has handed out all tasks in the current unfinished batch, the client waits a while before asking again – this waiting period is given by some function – the *back-off function* – ensuring that the network is not flooded with task requests from idle clients.

The client computers might have very different performance characteristics and tasks can have different complexity – thus some clients might finish their task much quicker than others. To begin with we consider clients with equal performance.

As each task is given by a chess position for two chess programs, it is impossible beforehand to know exactly how long the task will take to be processed. If we knew the time consumptions the problem would be somewhat different and more OR-oriented finding the optimal way of assigning the tasks to clients. However in practice the processing times will take on some distribution which we will use in our simulation system. This system has the Markov property: The distribution of all future states will only depend on the current state.

2.2 The System Model

The transformation of the system described above into a model goes like this:

¹The clients could be behind firewalls, which does not allow incoming connections.

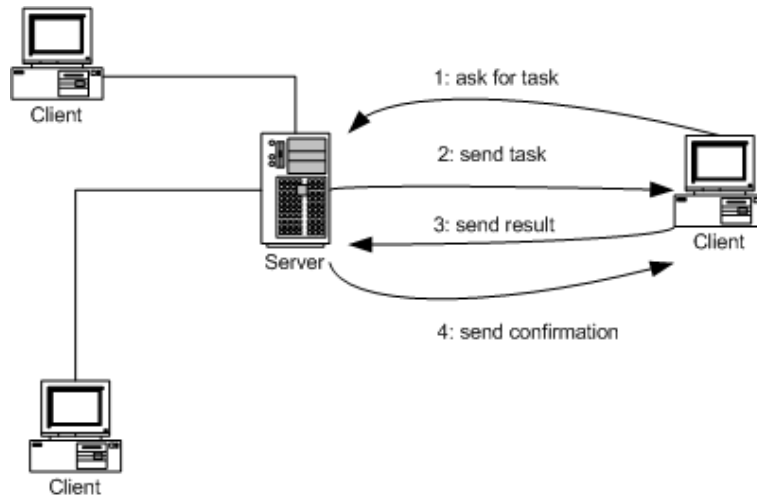


Figure 1: The principal layout of the client server system.

Our important concepts are the processing time of each task – which we assume can be described by a random variable X with a certain distribution g – and the back-off function for each client – which we determine by a function $f(n)$ of the current number of consecutive failed task requests n . That is, if a client has gotten “no available jobs” from the server four times in a row, it will wait $f(4)$ time units before asking again.

We assume that communication and server processing is instantaneous, and indeed that all other things than task processing time does not take any time.

Although clients might differ in processing power, we assume that they behave similarly for all tasks, so a client’s speed can be described by a single number, which is its speed compared to some reference machine. The scale of the distribution of the task processing times is chosen so that the real processing time on a machine with speed s is X/s .

Our model consists of

- n clients C_1, C_2, \dots, C_n with speeds s_1, s_2, \dots, s_n
- a common back-off function $f(n)$
- a sequence of batches with each m tasks of processing times sampled from the distribution g .

The state of our model consists of

- the number of tasks not yet processed in the current batch
- for each client the time until it will communicate again, and whether it will finish a task or ask again for a new task.
- for each client the number of times it has backed off since finishing its latest task.
- for each batch we store the accumulated time spent.

3 Determining Sample Distribution

The jobs in the system in mind were games of chess between two chess engines with different strategy parameters. In order to determine the distribution of the duration of such games, we timed 720 games of chess with settings equivalent of those in our real system. Figure 2 is a histogram of the results.

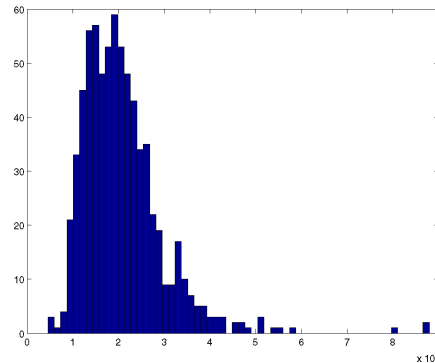


Figure 2: Histogram of duration (in ms) of chess games from 720 samples.

We try to model the data with a gamma and a log-normal distribution to ease sampling. This is done with Matlab's `gamfit` and `normfit` functions. We find, that data fits reasonably well with a log-normal distribution with mean 9.86 and standard deviation 0.389, see figure 3.

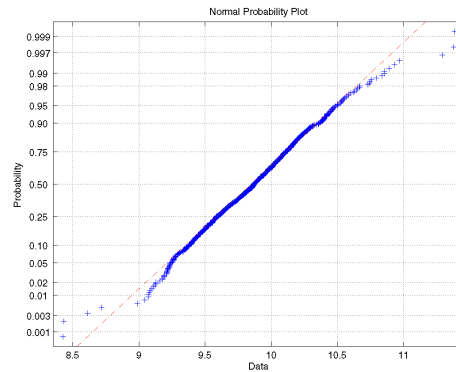


Figure 3: Normal plot of the log-transformed data.

It is doubtful that data is actually log-normal distributed, but it is a reasonable approximation in our case. Alternatively we could have used some kind of interpolated c.d.f. to sample from, e.g. linear interpolation.

4 Simulator implementation

Our simulator simulates the processing of a series of B batches with each T tasks performed on M clients. It's a state machine that maintains the state as described in section 2.2 and progresses using the event-by-event principle. The source is included as an appendix. The outline of the algorithm is presented in the following.

1. Initialize time of the next event for each client to $t = 0$, and set their state to “not running”. This means they all begin by quering the server for a new task.
2. As long as we haven't processed the whole sequence of batches, pick the next event as the lowest next-event-time of all the clients.
3. If this event was a completion of a task, update the statistics and let it ask the server for a new task.
4. If the event was the completion of the back-off time, ask the server for a task again.
5. If a task is available, set the client's next event to be “running”, and the event time to be a random sample from the processing time distribution (divided by the proper machine speed).
6. If no task is available, set the client's next event to be “not running”, increment its back-off count, and set the next event time to be calculated from the back-off function using the new back-off count.

5 Validation

This section is about the validity of the assumptions we have made – i.e. how well the model fits with the real world.

We have not been able to do proper validation by comparing real world data with simulated results, and we consider it out of scope to perform new real world experiments. Instead we have tried to analyse the assumptions made and their possible impact on the results obtained:

- We have neglected the possibility of a client failing. In practice this is solved by reassigning tasks given out more than x minutes ago. This was not a neglectable problem when the real experiment was conducted, because many of the used clients were dual-boot machines that were sometimes rebooted by a student – causing the task to be terminated without notifying the server. However, in common scenarios client failure could reasonably be neglected, because of the (hopefully) very low frequency of these.
- We have assumed zero delay on all network traffic. This seems like a reasonable assumption given the low amount of communication compared to the time of executing a job. Of course, this also assumes a waiting function that is large (in order of seconds) compared to the network delay, so there is no congestion.

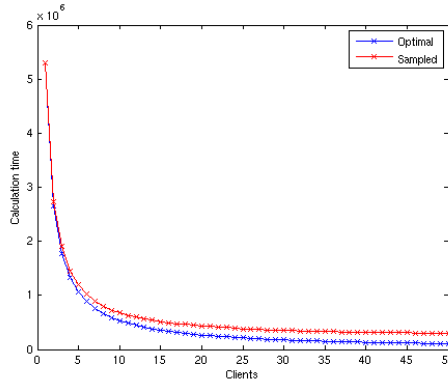


Figure 4: Calculation time as a function of the number of clients.

- We assume that the server never fails or gets overloaded and has no delay – this is also reasonable under the above made assumptions. The calculation of new parameters for each batch is practically instantaneous, so this is also ok.

6 Verification

In order to verify the implementation of the simulation model the program is executed using clients of equal speed with an exponential back-off function. We iterate on the number of clients from 1 to 50 while keeping the overall settings of the simulator. We execute 5 batches of 50 tasks in each step and consider the total time used and the number of rejections and tasks received for each client in the grid.

We wish to compare these values to the optimal case (i.e. no waiting time), so we sample 50 calculation times by means of the fitted lognormal distribution – using the sum divided by the proper number of clients as a measurement on how well the system would perform if the waiting time was completely eliminated. We plot these together with the output of the simulator (figure 4).

As expected the curve with the optimal calculation times is below the curve with the actual performances – except when using only 1 client where the waiting time in fact is eliminated. We also see that the time spent decreases as the number of clients increase as expected the calculation time is inversely proportional to the number of clients, and we observe that the curve is not linear.

For 1, 8, 34 and 50 clients we investigate the number of rejections and tasks received by each client (figure 5). As expected the single client situation results in 250 tasks and no rejections for the client. For the other three cases we see that the number of rejections and jobs are evenly distributed among the clients, which is expected as they are running at the same speed. As the amount of clients increase the number of jobs for each client decreases as anticipated.

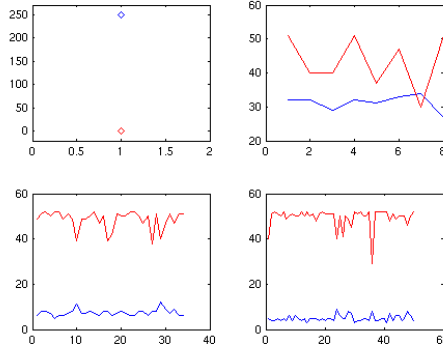


Figure 5: Rejections and tasks received for each client – respectively blue and red.

7 Results

In this section we present the results obtained. We show that the calculation time is almost inversely proportional to the number of clients used, that there is little difference between backoff functions, and finally that slower clients quickly becomes the bottleneck of the system.

7.1 Design of Experiments

As the behavior of the system is highly dependent on the number of clients used, k , all experiments are conducted by varying the number of clients and keeping the number of jobs pr. batch, N , constant. This constant is less important because characteristics naturally are about the same for equal ratios of k/N , so it is a question of granularity.

All experiments are conducted with a not too low number of batches to ensure achievement of steady state and to get lesser variance on the results. To reduce the variance further we use the same set of sampled job times when an experiment is repeated for a new number of clients. This is done either by resetting the random seed in Matlab or by using the same vector of sampled times to draw from.

7.2 Processing Times and the Back-off Function

We wish to investigate the influence of different back-off functions on the total processing time of a sequence of batches also looking at the amount of network traffic generated. All experiments are conducted iterating from 1 to 50 clients, all having a speed factor of 1. The results are plottet in the figures 6, 7, and 8.

7.3 Slow Computers

In this section we will try to examine some situations where the addition of a slow computer will actually slow the system down.

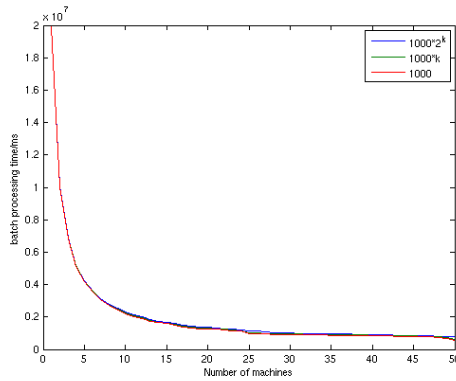


Figure 6: The total running time of 20 batches of 50 tasks using three different back-off functions. As it can be seen the running times are in the same order of magnitude, and vary mostly in the number of clients, roughly as the inverse number of clients.

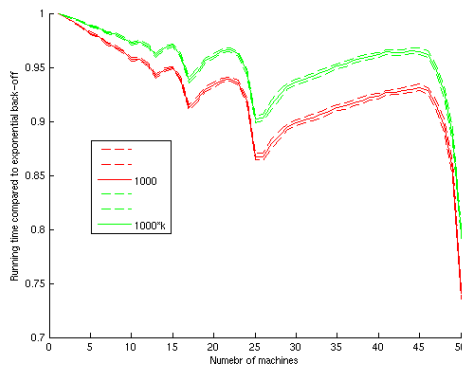


Figure 7: Comparing constant and linear back-off to exponential back-off we see that they are up to 25% faster. Interestingly we see the biggest gains around the number of clients being exact divisors of the number of tasks in the batch. This is most clear around 25 and 50 clients. On both sides of the two curves we show the 95% confidence interval of the mean value.

It seems obvious, that if the average task time of a slow client exceeds the average batch time (of the other clients), it will be a bad idea to add it to the system. Similarly, because a new batch cannot be started until all jobs have been completed, a slow computer can impede the start of a new batch if it gets one of the last jobs in the batch and thus slow down the overall performance. We now ask the question: given k clients with equal speed and N jobs pr. batch, can it pay off to add an extra client, with speed as e.g half of the all the others?

We simulate with a fixed batch size of $N = 200$ to find the computation time with and without the slow client for comparison. This experiment is repeated

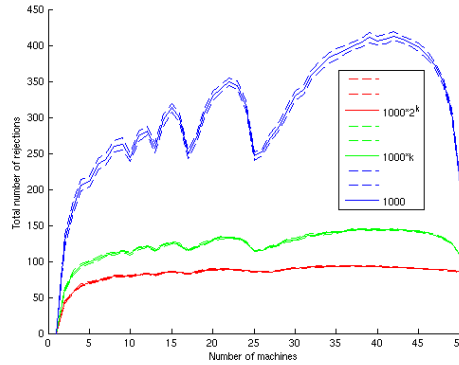


Figure 8: The total number of rejections on varying number of clients for different back-off functions. As expected exponential back-off gives the lowest number of rejections. For the constant back-off we can observe how the number of clients and number of tasks in a batch interact – there are clearly less rejections, when the number of clients is close to an exact divisor of the number of tasks. On both sides of the three curves we show the 95% confidence interval of the mean value.

for $k = 1$ to $N/4$ (this is the critical region) and for slow clients with speeds 0.1, 0.2, 0.33, 0.5, and 0.75 as opposed to all others with speed 1. Slow clients are simulated simply by dividing the sampled time with the speeds mentioned. The results are plotted in figure 9.

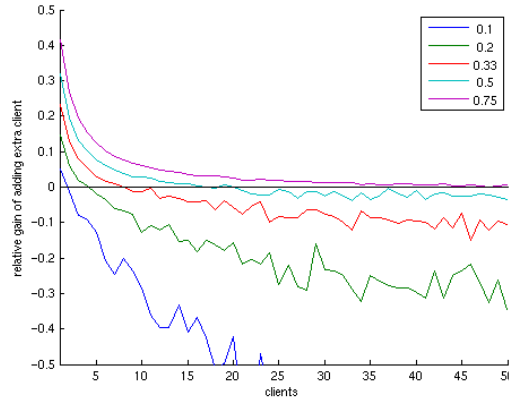


Figure 9: The gain of adding a slower computer to a grid of clients with equal speed.

From this we see, that one need not have that many clients before a slow computer becomes the bottleneck, and the answer to the particular asked question is around 15.

In the experiment above we have done no replicates, so we cannot state a

confidence interval around each of the plotted lines. However, we use 100 iterations (batches) in each subexperiment, which diminishes the error considerably, and the lines look fairly smooth with no intersections. If one wishes to investigate a specific scenario, then replicates and estimation of confidence intervals would be in order.

8 Conclusion

This report describes our simulation of a grid used to play games of chess between chess engines with the purpose of optimising an engine's evaluation function parameters. We have found that the duration of games under the given conditions can be described by a lognormal distribution, and we have shown how the grid can be represented as a state machine, which thus enables us to use Markov Chain Monte Carlo simulation of the system. We argued that our model is valid under certain conditions, and we have verified that the model behaves as expected. Finally we have designed and executed a set of experiments to reach our initial goals.

These experiments have shown that the computation time is roughly inversely proportional to the number of clients used, that a constant and linear back-off has little impact on the computation time as opposed to exponential, the difference being greatest when the number of clients is an exact divisor of the batch size. Finally we have shown that a slow computer very quickly becomes the bottleneck of the system, e.g. already at around 15 clients and a batch size of 200, it cannot pay off to add a client of half the speed.