The power of two paths in grid computing networks

Wouter ibens
University of Antwerp

May 5, 2012

Abstract

a

Contents

Introduction

Write at the end

1 Setup

We are using a ring-structured network of N nodes. Each node is connected to two neighbours, left and right. The purpose of these nodes is to process incoming jobs. When a node is busy while a job arrives, it must forward to another node. When a job has visited all nodes and none of them was found empty, the job is dropped.

Jobs have an arrival time, a length and optional metadata. They arrive at each node indepentently as a poisson process at rate λ. Their length is exponentially distributed with mean μ (unless otherwise noted, assume $\mu = 1$). Although each job has a length, this length may not known in advance. Finally, the metadata is optional and may be used by the nodes to pass information among the job (e.g. a list of visited nodes).

Nodes can use different algorithms to determine whereto a job will be forwarded. The performance of these algorithms is the main focus of this thesis. Different techniques will be discussed and simulated. Afterwards, some results of the simulation will be validated. Note that the cost of forwarding a job is neglected. Together with the presumption a job must visit each node before being dropped, this means a job arriving at any node will be processed if and only if at least one server is idle.

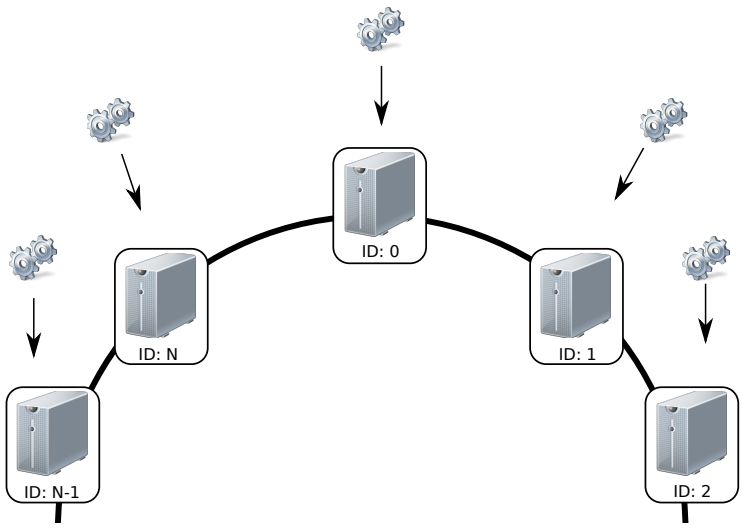

Figure 1: A ring structured network

The performance of a forwarding algorithms is measured by the average number of hops a job must visit before being executed. The goal of the algorithms is to minimize this number by spreading the load evenly along the ring.

1.1 Forwarding algorithms

Nodes that must forward a job must choose another node of the ring. Nodes have no information about other nodes, so is has no idea whether the node is idle or busy. The algorithms are grouped in two categories: forward to neighbour and forward anywhere. The first techniques allows a busy node to forward an incoming job to either its left or right neighbour, where the latter may forward these jobs to any node in the ring. Since the amount of dropped jobs is equal for each forwarding algorithm. These jobs will be ignored when computing the average number of hops.

1.2 Forward to neighbour

1.2.1 Forward right

A busy node using this technique will forward a job to its right neighbour. The job will keep travelling clockwise until an idle node is found, where it will be processed. This algorithms is used as base line in all further tests.

1.2.2 Left/Right forward

A variant to the previous algorithm is the Left/Right forward technique. Instead of forwarding each job to its right neighbour, a busy node will alternate the direction after forwarding such a job. To avoid a job coming back, this initial direction is saved in the job's metadata. Busy nodes receiving a job from another neighbour must forward it the same direction as specified in the job's metadata.

1.2.3 Random Left/Right forward with parameter p

This technique is a variant of the Left/Right forward algorithm. However, instead of alternating the direction for each new job, a node will forward a job to its right with probability p and to its left with probability $1 - p$. As the previous technique, the direction is saved in the job's metadata and subsequent nodes must maintain this direction when forwarding.

1.3 Forward anywhere

The ring strucure can be used in real networks, however in many cases the ring is no more than a virtual overlay over another structure (e.g. the internet). In these networks each node is able to connect to each other node. In these networks, other forwarding algorithms can be used.

1.3.1 Random unvisited

When nodes aply this technique, a forwarding node will save its ID in the jobs metadata. When another busy node must forward the job it will choose a random node that has not been visited before.

1.3.2 Round Robin unvisited

1.3.3 Coprime offset

Another technique uses coprimes. When the ring initialises, a list of numbers is generated. Every number from 1 to the size of the ring minus one, that is coprime to the size of the ring is saved in this list. For example, when the ring has 10 nodes, the list contains $\{1, 3, 7, 9\}$. When a busy node receives a job, the next number from the list is choosen and saved in the jobs metadata, the job will travel in steps equal to this number.

1.3.4 Random Coprime offset

This technique is analogue to the previous one. But instead of taking the next number from the list, a random value from this list is choosen.

2 Simulation

The continuous time simulator is written in C++ wihtout any requirements except the STL. It's behavour can be controlled using a command line interface. The source code of the code can be found in appendix A. The program can be used using the options given in the usage description given below.

```
Usage: -r -s long -j double -a double -n long -p long -l long -t
    long -h type
        -r      Random seed
        -s      Set seed                          (default: 0)
        -j      Job length                        (default: 1.0)
        -a      Interarrival time                 (default: 1.0)
        -n      Ring size                         (default: 100)
        -p      Print progress interval (default: -1 - disabled)
        -l      Simulation length                 (default: 3600)
        -t      Repetition                (default: 1)
        -h      Print this help
         type   right | switch | randswitch | evenswitch | prime |
            randprime | randunvisited | totop
```

Listing 1: Simulator usage description

All results are obtained using a ring size of 100 and random seeds.

2.1 Measure

The goal is to destinguish algorithms on how well they distribute the incoming jobs. To achieve this we will compute how much a node is forwarded within the ring. We will only take jobs into account that could be succesfully completed to make the results more clear.

One should note that the success rate measure is useless here since every job will be executed as long as any node in the system is idle, which is just the Erlang loss probability.

It is clear that when the load approaches 0, the probability that a node is busy will also approach 0 and the average number of hops will therefor also approach 0. On the other hand, when the load approaches ∞, the node's probability to be busy will approach 1 and therefor the number of hops will be $N-1$ and the job will fail. A system with load > 1 is called an overloaded system.

The baseline for these results will be the Forward Right technique, all other techniques will be depicted relative to the Forward Right baseline result. The absolute performance can be seen in figure 2.

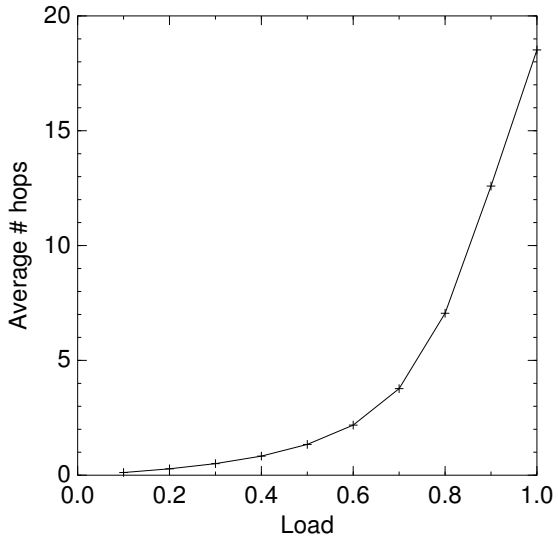

Figure 2: The Forward Right baseline result

2.2 Forward to neighbour

2.2.1 Left/Right Forward

It is intuitivly clear that alternating the direction of incoming jobs distributes the load slightly better than keeping the same direction. The performance measure is better than the baseline result for every load level (figure 3). But the improvement is most clear below 0.8.

2.2.2 Random Left/Right forward with parameter p

For $p = 0.5$, this technique is similar to the previous one. Thus one would expect similar results. However, the results (figure 4) are even worse than our baseline results.

This results raises the question how much the parameter influences this result. Since for $p = 0$, the algorithm is equal to the Forward Right technique and for $p = 0.5$ the results are measurable worse. We will look further into this in section

2.2.3 Position-dependant forwarding

This technique groups nodes in virtual clusters. When a job arrives in a node and that node is busy, the job will be forwarded to the other node in the cluster. Jobs leaving a cluster will do this in a random direction ($p = 0.5$). Since the load is concentrated per cluster instead of being distributed over the whole system,

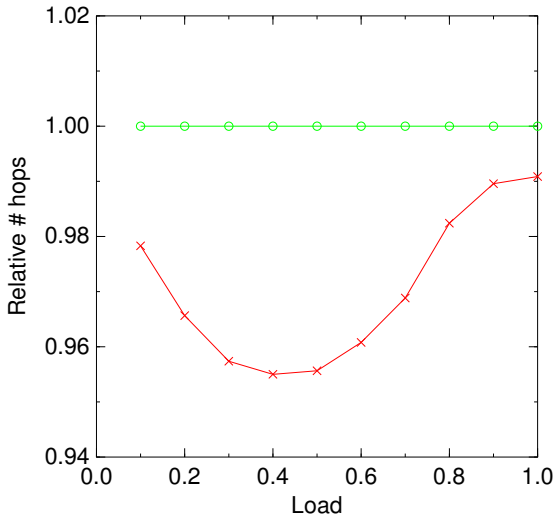

Figure 3: Left/Right

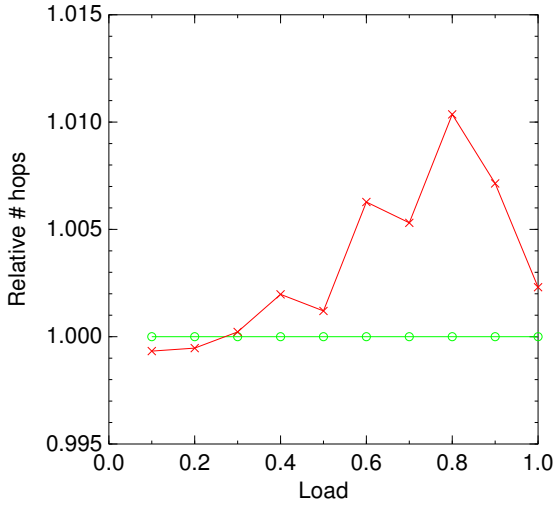

Figure 4: Random Left/Right forward with parameter 0.5

this technique performs worse than other techniques The results are represented in figure 5.

2.3 Forward anywhere

2.3.1 Random unvisited

This algorithm in the most straight forward and is the best performing from any of these techniques. However, it should be noted that each visited node must be stored into the job's metadata.

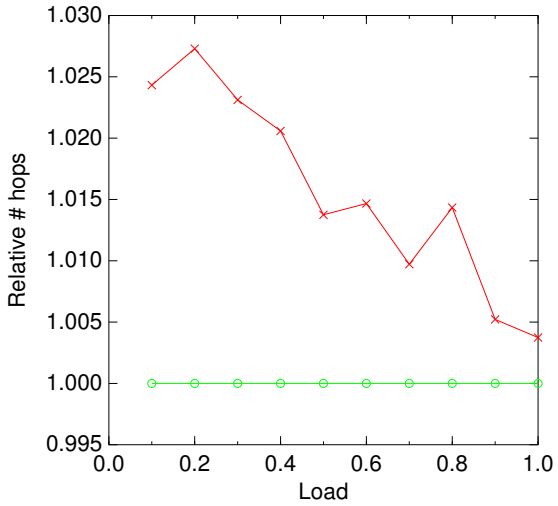

Figure 5: Position-dependant forwarding

3 Numerical Validation

To validate the results obtained in the previous section, we modelled the scheduling-techniques into Markov Chains. Using the steady state distribution of these chains, we can derive the average number of hops and the average loss. For N nodes in a ring, the markov chain consists of 2^N number of states, where the bits represent whether a server is busy or idle. To optimize the computation time and memory requirements, we used sparse matrixes for the validation. The validation code is written in MATLAB.

3.1 Forward Right

As for this technique, the matrix representing the Markov chain is easily determined. The example given below is for a ring of 3 nodes. For convenience, the states are representes by their binary form.

$$Q = \begin{array}{c} \\ 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \begin{array}{c} \begin{array}{cccccccc} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{array} \\ \begin{bmatrix} -3\lambda & \lambda & \lambda & 0 & \lambda & 0 & 0 & 0 \\ \mu & -3\lambda-\mu & 0 & \lambda & 0 & 2\lambda & 0 & 0 \\ \mu & 0 & -3\lambda-\mu & 2\lambda & 0 & 0 & \lambda & 0 \\ 0 & \mu & \mu & -3\lambda-2\mu & 0 & 0 & 0 & 3\lambda \\ \mu & 0 & 0 & 0 & -3\lambda-\mu & \lambda & 2\lambda & 0 \\ 0 & \mu & 0 & 0 & \mu & -3\lambda-2\mu & 0 & 3\lambda \\ 0 & 0 & \mu & 0 & \mu & 0 & -3\lambda-2\mu & 3\lambda \\ 0 & 0 & 0 & \mu & 0 & \mu & \mu & -3\mu \end{bmatrix} \end{array}$$

Analogue to the simulation section, this method will be the baseline result in our other results.

3.2 Random Left/Right forward with parameter p

This matrix is very similar to the one above. But we need to take into account the parameters p and $1-p$ instead of 1 and 0.

$$Q = \begin{array}{c} \\ 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \begin{array}{c} \begin{array}{cccccccc} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{array} \\ \begin{bmatrix} -3\lambda & \lambda & \lambda & 0 & \lambda & 0 & 0 & 0 \\ \mu & -3\lambda-\mu & 0 & (2-p)\lambda & 0 & (1+p)\lambda & 0 & 0 \\ \mu & 0 & -3\lambda-\mu & (1+p)\lambda & 0 & 0 & (2-p)\lambda & 0 \\ 0 & \mu & \mu & -3\lambda-2\mu & 0 & 0 & 0 & 3\lambda \\ \mu & 0 & 0 & 0 & -3\lambda-\mu & (2-p)\lambda & (1+p)\lambda & 0 \\ 0 & \mu & 0 & 0 & \mu & -3\lambda-2\mu & 0 & 3\lambda \\ 0 & 0 & \mu & 0 & \mu & 0 & -3\lambda-2\mu & 3\lambda \\ 0 & 0 & 0 & \mu & 0 & \mu & \mu & -3\mu \end{bmatrix} \end{array}$$

The lumped matrix of Q is equal to the lumped matrix of the example above (3.5), i.e. the matrix defines the exact same problem. However, for $N > 6$ the matrices and so the results of the steady state distribution begin to differ.

3.3 Random Coprime offset

Modelling this technique yields different results for various ring sizes. The performance of this algorithm is very dependant on the number of coprimes that can be used. This technique yields the same results as Forward Right for ring sizes of up 4. For $N = 3$, the matrix Q is identical to Random Left/Right forward with parameter 0.5, as the coprimes of 3 are 1 and 2. Which means forwarding a job left or right, both with the same probability.

3.4 Random Unvisited

This problem can be modelled much more efficiently than the techniques above. Since the next node is choosen randomly, the information we need to save consists only of the number of servers which are currently busy. This problem is analogue to modelling an Erlang-B loss system. The number of states in this Markov Chain is linear to N and is much more dense. For $N = 3$, the matrix is given below.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} -3\lambda & 3\lambda & 0 & 0 \\ \mu & -3\lambda-\mu & 3\lambda & 0 \\ 0 & \mu & -3\lambda-\mu & 3\lambda \\ 0 & 0 & \mu & -\mu \end{bmatrix} \end{array}$$

3.5 Lumped states

Except for Random Unvisited, each discribed technique is modelled into a matrix with N^2 states. However, may of these states are redundant: for example, for $N = 3$ the states 001, 010 and 100 all represent one of the nodes being busy. For states where multiple nodes are busy, the space between these servers is

critical information, therefor states are lumped when rotating the bits of one state becomes equal to another state. The states below are analogue and can be lumped:

`001101 = 011010 = 110100 = 101001 = 010011 = 100110`

The number of states in a lumped MC is $\frac{1}{N}\sum_{d|N}(2^{N/d} \cdot phi(d))$ with $phi(d) = d \cdot \prod_{p|d, p \text{ is prime}}(1 - \frac{1}{p})$ [1]. This result greatly reduces the number of states, however its complexity is still non-polynomial.

The example model in 3.1 can be lumped into the following Markov Chain:

$$Q = \begin{array}{c} \\ 000 \\ 001 \\ 011 \\ 111 \end{array} \begin{array}{c} \begin{array}{cccc} 000 & 001 & 011 & 111 \end{array} \\ \begin{bmatrix} -3\lambda & 3\lambda & 0 & 0 \\ \mu & 3\lambda - \mu & 3\lambda & 0 \\ 0 & 2\mu & 3\lambda - 2\mu & 3\lambda \\ 0 & 0 & 3\mu & -3\mu \end{bmatrix} \end{array}$$

3.6 Equivalent techniques

Lumping states of a Markov Chain produces an equivalent Markov Chain. This can be used to prove some forwarding techniques are equal up to a certain N.

4 Conclusion

Which techniques work best in which environments? Why? Runner up? Why do some techniques don't work as expected?

Acknowledgements

Thanks to everyone

References

[1] The Online Encyclopedia of Integer Sequences. *A000031*. June 2009. URL: `https://oeis.org/A000031`.

A Simulator source code

dit is de inhoud

B MATLAB Numerical evaluation code

een appendix