Masterthesis:
The power of two paths in grid computing networks

Wouter ibens
University of Antwerp

February 28, 2012

Promotor:
Benny Van Houdt

Abstract

a

Contents

Introduction

Write at the end

1 Setup

Explain the setup, the goals, the reasons, explain the balancing-techniques Insert a picture of nodes in a ring

Imagine a ring-structured distributed system, consisting of several nodes, each connected its two neighbours. Nodes may have x servers, meaning they can process at most x jobs at a time. External jobs will arrive at each node, independent of eachother. Busy nodes will pass an incoming job on to another node using a specified algorithm. This setup is fixed during the whole thesis. Our goal is to examine how well given algorithms distribute the load of the system. It is measured using the average number of times a job must be forwarded before is can be executed. Jobs that are forwarded to any node in the system are discarded. We assume the jobs have a Poisson arrival rate with parameter λ, the service time is exponental with parameter μ, and forwarding a job takes no time at all. Unless otherwise noted, we assume $\mu = 1$.

As performance measure in this thesis is the average number of times a job visits a busy node. Since the loss rate is the same for every technique (a job will visit every node before being dropped), we will only take into account the jobs that were finished.

1.1 Techniques

Various forwarding rules will be discusses, they can be divided into two groups: 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 system. All techniques discussed in this thesis will make sure a job will visit every node at least once before returning to their originating node. Note however, that nodes that would be forwarded to their originating node will be dropped instead.

1.2 Forward to neighbour

Following techniques only allow a busy node to forward a job to its left or right neighbour.

1.2.1 Forward right

This is the most straight-forward technique, a busy node will always forward a job to its right node. In a saturated system, a job will visit the ring in a clockwise direction before being dropped.

1.2.2 Left/Right forward

When using Left/Right forward, a busy node that receives a new job will forward the job either left or right. The direction is saved in the job metadata, following

busy nodes will forward the node in the same direction. The direction will alternate for every new job that cannot be served.

1.2.3 Random Left/Right forward with parameter p

As the Left/Right forward technique, the first node will push the job in a certain direction, instead of alternating the direction. De direction is right with probability p and left with probability $1 - p$.

1.2.4 Position-dependant forward

Another technique consists of always choosing the same direction at the incoming node, but alternating that direction for every node. This technique gives a certain node ID 0, its right neighbour ID 1, and so one, until every node in the ring has an ID. When a node its ID is even, it will always forward new jobs right, if its ID is odd, if will forward them left.

1.3 Forward anywhere

When the ring-formation of the system is just a virtual overlay, but the nodes are actually connected in a less structured network like the internet, instead of only its neighbours, a node can reach any other node in the distributed system. Other techniques can be used to distribute incoming jobs elsewhere.

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

As part of this thesis, a simulator is developed to easily test the discribed techniques. The code can be found at `http://code.google.com/p/powerofpaths/`.

//TODO update when changed

The usage is described 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)
        -c      Processing units per node         (default: 1)
        -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 |
```

We used the simulator to test the various forwarding techniques.

//TODO How does the simulator work, what are its results

Below, we will show the results we discovered using the various techniques described in section 1.1. We will plot the results against the average load of the system. Every result will be averaged over 10 random runs. Unless otherwise noted, the ring size is 100 and each node has one server.

2.1 Forward to neighbour

2.1.1 Forward right

2.2 Forward anywhere

3 Numerical Validation

How is it validated, give an complete example, compare results

4 Conclusion

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

Acknowledgements

Thanks to everyone

References

Insert them

Appendices

A Source code

a