

# Modelling Network Performance on the IBM SP RS/6000

## CS 281A Final Project

Kaushik Datta (kdatta@cs.berkeley.edu)

Due: December 17, 2004

## 1 Introduction

Most large-scale scientific applications are now run on large supercomputers with many processors. While the workload is distributed, an important new concern that needs to be addressed is the network that allows these processors to communicate. Depending on the application, network performance can make a substantial difference in the overall running time.

This report looks to produce a preliminary model the performance of the IBM SP RS/6000, a typical supercomputer with many hundreds of nodes. Four key parameters were chosen as variables, including the number of nodes and message size. We tried three different models, the first two based on GLIMs (Generalized Linear Models), and the third one less so. We then sought to compare the quality of these models by plotting the distribution of relative errors for each model.

## 2 Problem Setup

We wanted to create a communication benchmark that could be used in modelling the performance of common scientific kernels (e.g. FFT, conjugate gradient, and multigrid). Common communication patterns in these kernels include nearest neighbor communication and all-to-all communication.

Our benchmark tries to mimic these patterns by pairing each node with a partner so that the processors on one node only send messages with the processors on its node pair. This is nearest neighbor communication, so we can model that pattern. This can also be considered one step in all-to-all communication, so that pattern can be modelled as well.

### 2.1 Platform

The IBM SP RS/6000 system that was used (seaborg.nersc.gov) is a distributed memory computer with 380 compute nodes. Each Shared Memory Processor (SMP) node contains 16 375 MHz Power3 processors and two network adapter cards for inter-node communication.

## 2.2 Variables

There were five variables for this benchmark:

1.  $c$  = unidirectional (0) or bidirectional (1) communication
2.  $n$  = number of nodes (2-8)
3.  $p$  = number of processes/node (1-16)
4.  $t$  = number of threads/process (1-16)
5.  $m$  = message size in Bytes ( $2^7$ - $2^{26}$  B)

We need to include these input variables in our model to predict our response variable, time.

## 2.3 Data Collection

For unidirectional communication, only one node in each node pair sends messages, while the other node in the pair receives the messages. For bidirectional communication, both nodes in the pair send their messages simultaneously, and then wait for the message from its pair. In order to time these events, the timer is started immediately before the nodes begin communicating, and the timer is stopped after the last node completes receiving its message.

In addition, since each node contains 16 processors,  $p * t \leq 16$ . This does limit our data collection, but this is a relatively easy hurdle to overcome.

A larger hurdle presents itself in the range of message sizes that were sent. In order to observe both the latency and the bandwidth of the network, very small as well as very large messages needed to be sampled. Thus, starting at  $2^7$  B, message sizes were doubled up to a maximum size of  $2^{26}$  B. While this provides us a good network profile, the large difference in message sizes does cause some problems with using GLIMs. This will be discussed in further detail later.

In order to reduce noise in the data, each data point was run 10 times, and the median of these values was used in the final data set.

## 3 Data

To get an intuition of how the data looks, the data was plotted as surface graphs so we can vary two of the variables at once. In Figure 1,  $n$  and  $p$  are varied while holding  $t$  and  $m$  constant. Outside the point (8, 16), which seems to be an outlier, the time seems to be linear in both  $n$  and  $p$ .

In Figure 2,  $n$  and  $t$  are varied. It looks like the value of  $n$  makes almost no difference in the time, while  $t$  looks to be directly proportional to time.

Figure 3 varies  $p$  and  $t$ , but since we have the restriction that  $p * t \leq 16$ , the graph is very limited. Nonetheless, it seems to show that the time is linear in  $p$  and  $t$ .

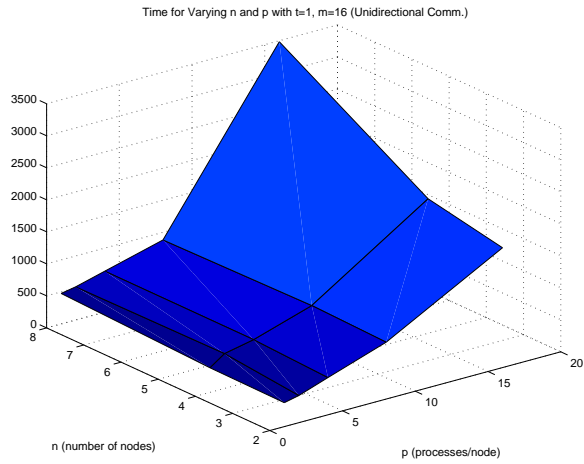


Figure 1: Varying n and p

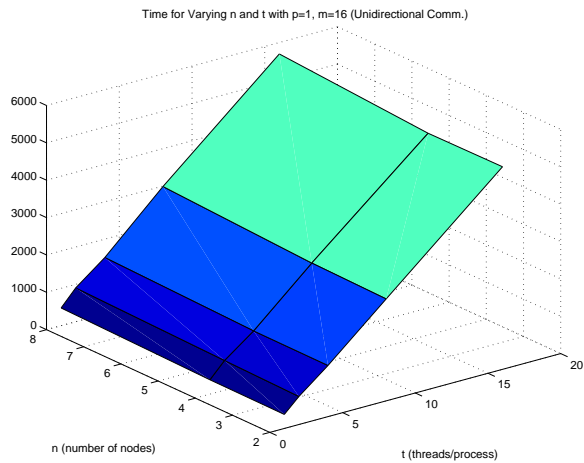


Figure 2: Varying n and t

The final data figure, Figure 4, varies t and m. This is a log-log graph since m ranges through such a large range of values. It shows at which message sizes latency dominates (flat graph) and which sizes bandwidth dominates (rising graph).

All these figures seem to indicate that, at least as an approximation, time is linear in n, p, t, and m.

## 4 Models

There were three models that used to fit the data. The first two attempted to use Generalized Linear Models (GLIMs), while the third uses GLIMs in a more piecewise fashion. In a GLIM, the data is assumed to be linearly dependent on some function of each of its variables. We will assume that our output variable time ( $y$ ) is drawn from the Gaussian distribution. Then we can say:

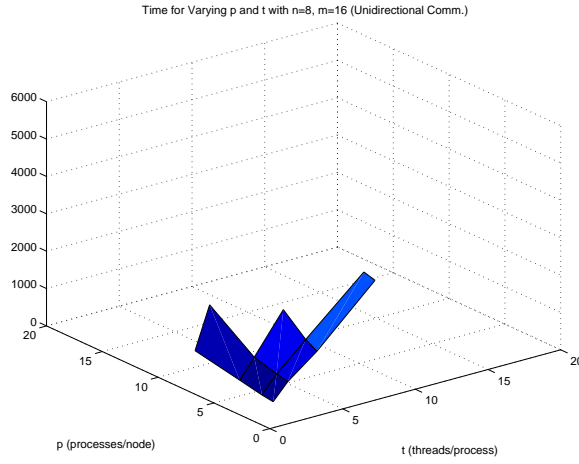


Figure 3: Varying p and t

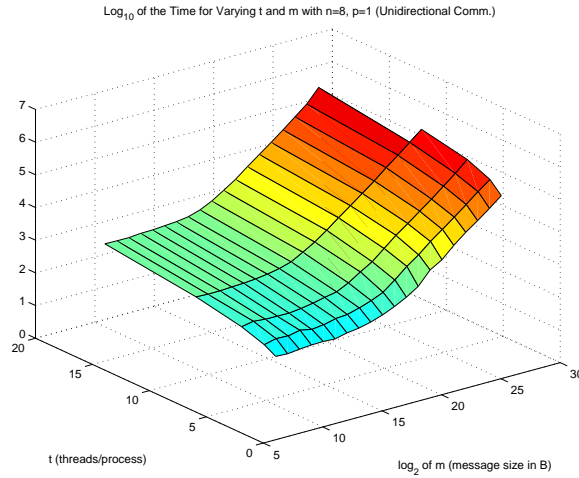


Figure 4: Varying t and m

$$E(Y_i) = \mu_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_5 x_{i5}; \quad Y_i \sim N(\mu_i, \sigma^2) \quad (1)$$

where the  $\beta$ 's are the parameters we are trying to determine.

However, our data is not of this form yet. Our data can be written in the approximate form:

$$y_i = \beta_0 + (\beta_1 n + \beta_2)(\beta_3 p + \beta_4)(\beta_5 t + \beta_6)(\beta_7 m + \beta_8) \quad (2)$$

Examining our data more closely, we can make some assumptions to simplify the above equation. First,  $\beta_0$  can be set to zero, since the data approximately goes through the origin. Second, the data doesn't seem to depend on  $n$ , since we only scale up to 8 nodes. Thus, we will also set  $\beta_1$  to zero. Finally, since  $t$  seems to be directly proportional to  $y$ , we will also set  $\beta_6$  to zero. So:

$$y_i = \beta_2\beta_5t(\beta_3p + \beta_4)(\beta_7m + \beta_8) \quad (3)$$

We will use this equation as the starting point for our first two models.

#### 4.1 Linear Model

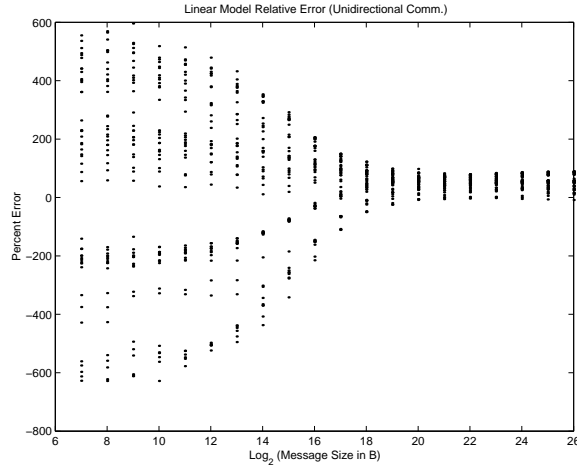


Figure 5: Linear Model Relative Error (Unidirectional Communication)

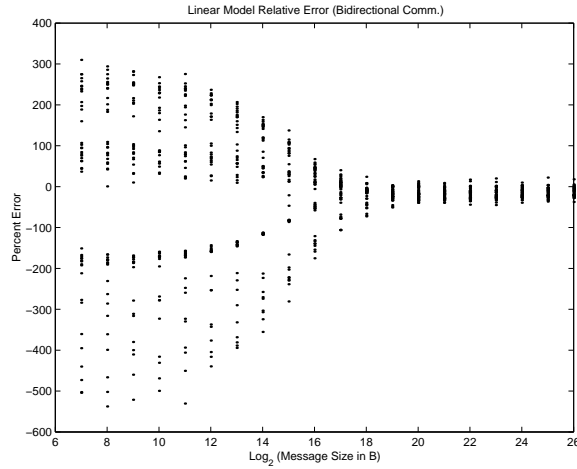


Figure 6: Linear Model Relative Error (Bidirectional Communication)

By multiplying out the above equation, we can make it a linear function of functions of our variables:

$$y_i = \beta'_1mpt + \beta'_2mt + \beta'_3pt + \beta'_4t \quad (4)$$

We then performed linear regression (using least squares) to obtain the  $\beta'$  parameters. If we keep  $m$  in Bytes, the fit is extremely poor, since the first two terms of the above equation

overwhelm the last two. So in order to have similar weights for each of the parameters,  $m$  was converted to MB. As seen in Figures 5 and 6, this model is very poor for small and medium message sizes, but does perform adequately for the largest sizes.

This effect is due to the large range of message sizes we are modelling. This model would be adequate if all the message sizes were within an order of magnitude. But since they are not, and we are doing unweighted least squares fitting, the largest message sizes are weighted much more than the smallest sizes, and thus our results.

## 4.2 Log Model

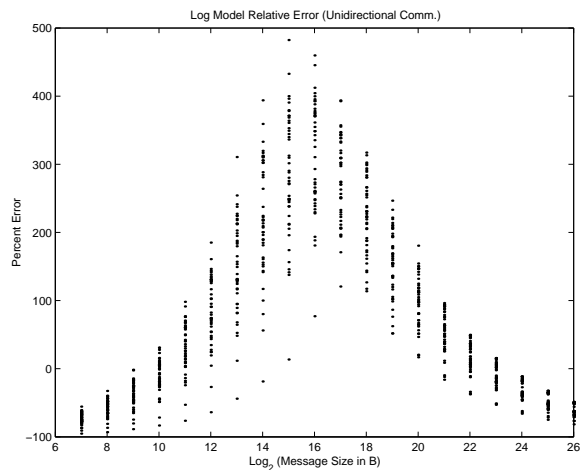


Figure 7: Log Model Relative Error (Unidirectional Communication)

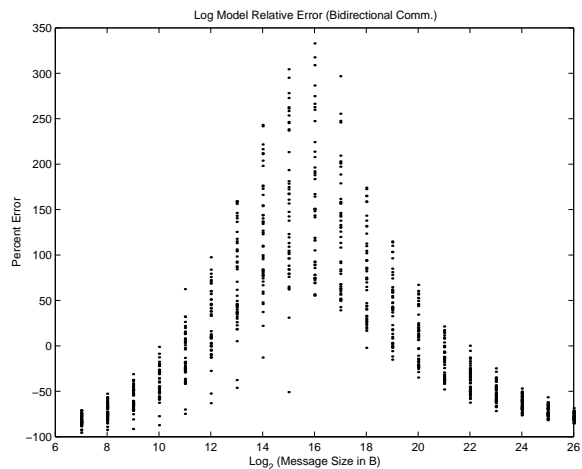


Figure 8: Log Model Relative Error (Bidirectional Communication)

A better approach would be to use the log of the message size, since the numbers are more tractable and least squares should be valid. However, simply taking the log of both

sides of Eq. (3) does not produce a linear equation. So with a few more assumptions, we can linearize it:

$$\log y_i = \beta_0'' + \beta_1'' \log m + \beta_2'' \log p + \beta_3'' \log t \quad (5)$$

Again, least squares was implemented to obtain the  $\beta''$  parameters. Looking at Figures 7 and 8, this produces better results than the linear model, but it only does well at the middle of each half of the message size spectrum. This model may not have fit as well as expected because of the many assumptions that were used to produce a linear equation. All of them may not have been safe to assume, causing errors in the log model.

### 4.3 Piecewise Model

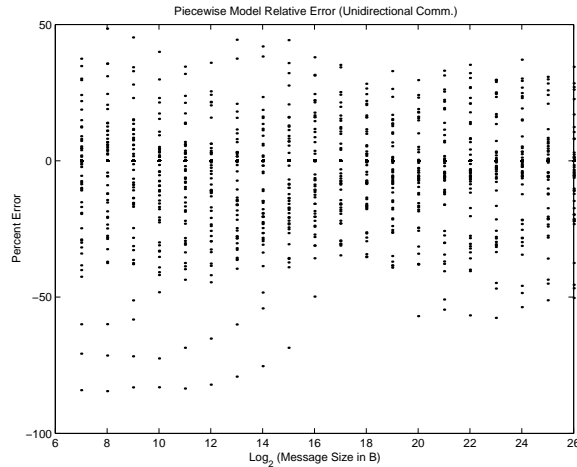


Figure 9: Piecewise Model Relative Error (Unidirectional Communication)

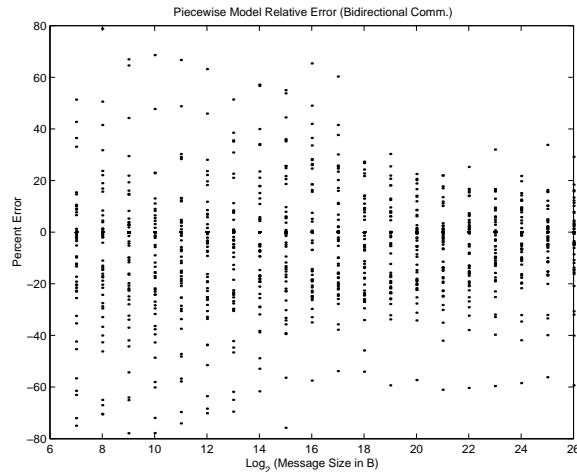


Figure 10: Piecewise Model Relative Error (Bidirectional Communication)

Our final model does not try to fit all the data at once. Instead, we integrate each variable into the model one step at a time. The obvious variable to start this model is  $m$ , since typically an  $\alpha$ - $\beta$  model is used to model network performance, and this model is based on message size. So first we performed weighted linear regression on each  $(n, p, t)$ -tuple, while varying  $m$ . The regression was weighted so that smaller message sizes were weighted equally with larger message sizes. This would produce an  $\alpha$  (latency) and  $\beta$  (inverse bandwidth) for each  $(n, p, t)$ -tuple. Then we integrated  $t$  into the model so that it is used to generate  $\alpha$  and  $\beta$ . In a similar way,  $p$  was then integrated into the model. As seen in Figures 9 and 10, this model seems to do the best.

## 5 Conclusion

The piecewise model works best because we adequately addressed the issue of drastically different message sizes. The other models would have worked equally well had it not been for this issue.