Network-aware Distributed Computing: A Case Study

Hongsuda Tangmunarunkit and Peter Steenkiste

School of Computer Science
Carnegie Mellon University

Abstract. The development of network-aware applications, i.e. applications that dynamically adapt to network conditions, has had some success in the domain of multimedia applications, but progress has been very slow for distributed computing applications. The reason is that the relationship between application performance and network performance is typically more complex for that class of applications, making adaptation difficult. In this paper we introduce two adaptation methods for distributed computing applications, one based on a performance model and another based on balancing computation and communication time. We illustrate the two methods using a simple distributed application (matrix multiply) and compare their performance. We show that both methods can correctly estimate the best number of nodes to use on our testbed. We also show that both methods have weaknesses. Model-based adaptation requires an accurate performance model and is sensitive to errors in measurements of the system parameters. The ratio-based method is more robust but less general.

1 Introduction

Over the last decade, we have seen increasing interest in using collections of workstations and PCs connected by networks as a distributed computing platform. While these "distributed multicomputers" usually have communication performance that is inferior to that of special-purpose supercomputers (e.g. Intel Paragon, Cray T3E), for a wide class of applications, multicomputers offer a very attractive performance/cost ratio. However, distributed multicomputers introduce significant new programming challenges. While the characteristics of special-purpose supercomputers are static and well-defined, the characteristics of multicomputers can be highly variable. For example, both node and network properties can be widely different from one invocation to the next. Moreover, unless the multicomputer is used in batch mode (e.g. a workstation cluster in a supercomputer center), the performance characteristics can change during an invocation as a result of resource sharing with other users. As a result, the application will have to adapt to the status of the system throughout the execution.

This research was supported in part by the Advanced Research Projects Agency/ITO monitored by NRaD under contract N66001-96-C-8528.

Hongsuda Tangmunarunkit is currently a graduate student at USC.

Changes in the availability of resources create a problem for distributed applications, since the optimal (or even a good) mapping of the application depends on the system and its status. For example, the power of the end-points (e.g. FLOPS rating) relative to the performance of the network (e.g. sustainable Mbs) often has an impact on how many nodes should be used, or on the optimal pipeline depth for pipelined computations [11]. Since applications can be invoked on systems with widely different characteristics, it is important that they can *adapt* to a specific configuration, i.e. they should be able to determine the optimal way of distributing themselves and the distribution may have to be changed during execution.

The problem of adaptation has received a fair bit of attention, especially in the area of multimedia, e.g. network-aware streaming of audio [6, 2] and video [5, 14, 7, 9] data. The performance of distributed computing applications tends to be much harder to characterize, making it more difficult to make them network-aware. In this paper we introduce two different mechanisms for making an application network-aware, one based on a performance model of the application and another based on the observed communication to computation ratio of the application. We compare how well the methods perform for a very simple problem: determining the optimal number of nodes for a distributed matrix-matrix multiply. The ultimate goal is to develop runtime systems that support adaptation to network conditions at startup and during execution, based on runtime measurements and application input.

The remainder of this paper is organized as follows. In Sections 2 and 3 we describe two different methods for application-specific adaptation. In Section 4 we show how the methods can be used to estimate the optimal number of nodes for simple matrix multiply, and we compare the performance of both approaches. We discuss related work and summarize in Sections 5 and 6. This paper is an extension of the first author's undergraduate thesis [13].

2 Model-based adaptation

Model-based adaptation is based on a performance model of the application that expresses performance as a function of system and application parameters. For example, the execution time could be expressed as a function of input data size, number of compute nodes, and the performance characteristics of the nodes and network. By taking the first derivative with respect to a particular parameter, e.g. the number of nodes, it is then possible to derive the value of that parameter that will optimize performance, as a function of the other parameters. Applications that use model-based adaptation collect information about system characteristics when they start up, either by querying the system through a special interface [4], or by running a small set of benchmarks. The system parameters are then plugged into the system model, which yields the method of distribution (e.g. the number of nodes) that optimizes performance for this particular system configuration. This method has been used successfully for a small but diverse number of applications [11, 10].

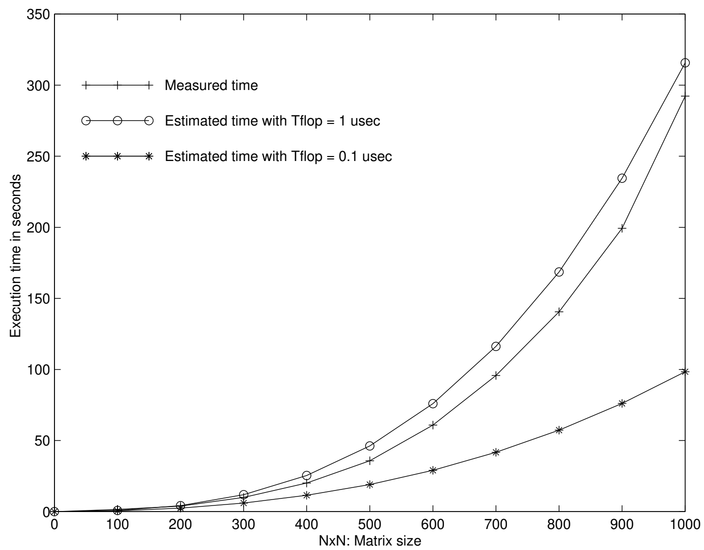

Fig. 1. Impact of T_{flop} on model accuracy (4 nodes)

The success of model-based adaptation depends on having both an accurate performance model and accurate measurements of system characteristics. The latter turns out to be more challenging than one would expect. Figure 1, for example, shows the fit between the measured performance for a distributed matrix multiply and a performance model (described in Section 4) on four nodes for different estimates of T_{flop}, the execution time for one floating point operation on the hosts (inverse of the FLOPS rating). Any of the T_{flop} in the graph can easily be "justified" by picking the right benchmark. T_{flop} is highly variable because of cache effects: benchmarks with small data sizes yield a much lower T_{flop} than benchmarks with larger data sizes and low cache locality.

The fact that the basic performance parameters of a system (such as T_{flop}) are potentially highly application specific clearly complicates the use of performance models. The characterization of the system is not a generic task that can be done once, but has to be repeated for every application. Not surprisingly, the most accurate benchmark to measure system parameters is the application itself, e.g. one ends up doing something similar to profiling. Figure 2 shows for example that the performance model for matrix multiply is quite accurate if the T_{flop} parameter is measured using a sequential matrix multiply.

3 Communication-to-computation ratio

One of the more critical system features affecting parallelization is the compute power of the end points relative to the communication performance of the network. In general, the faster the network (or the slower the nodes), the easier it is to use large numbers of nodes effectively, while a slower network (or faster

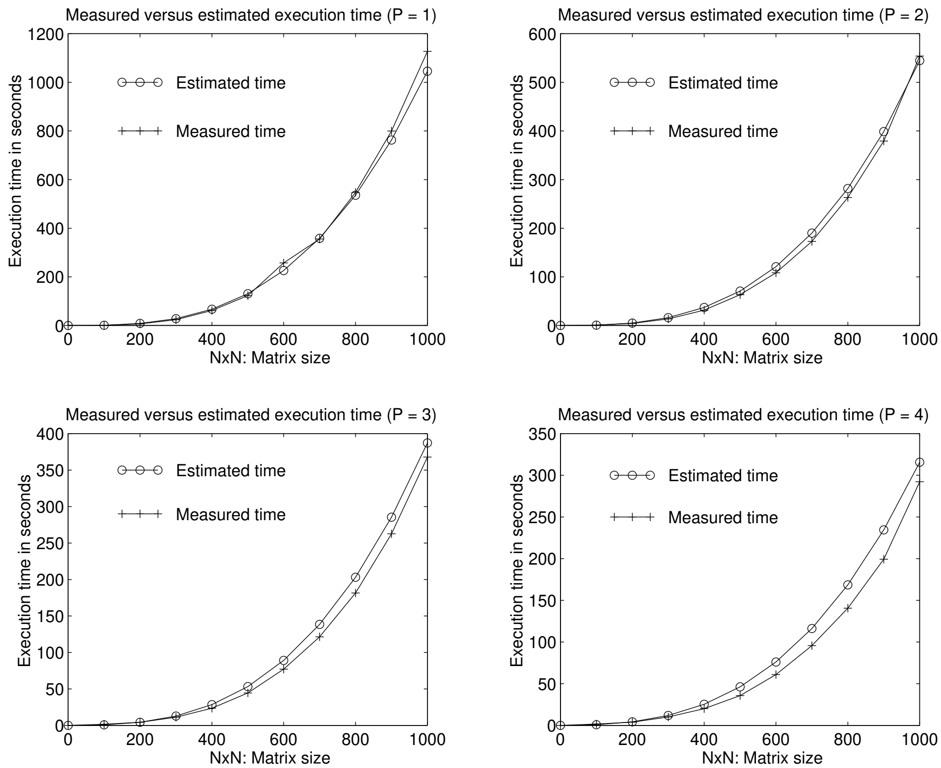

Fig. 2. Model accuracy relative as a function of the number of nodes

compute nodes) typically limits the degree of useful parallelism in applications. Many parallel applications consist of a sequence of computation and communication phases, and a slower network will make the communication phases more expensive while faster nodes will speed up the computational phases. One would expect that in a correctly distributed application, there will be some balance between the amount of time spent on communication and computation. In the remainder of this section we characterize this balance more precisely for one particular class of applications.

Let us assume a class of parallel applications in which the execution time can be represented as

$$T = \frac{T_{comp}}{P^p} + P^m * T_{comm} \tag{1}$$

where P is the number of nodes and T_{comm} and T_{comp} represent base computation and communication times. The first term indicates that the time spent on computation will decrease with the number of nodes while the second shows that the communication time increases. The parameters p and m are application specific, while m may also depend on the network or interconnect topology.

We would now like to calculate the ratio of the communication and computation times for P_{opt}, the number of nodes that minimizes the execution time T.

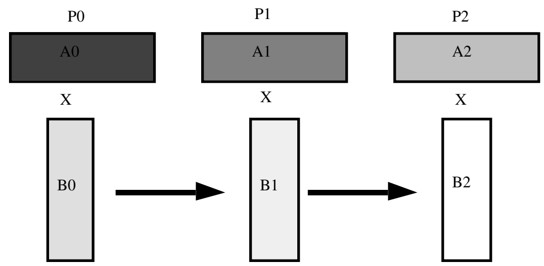

Fig. 3. Distributed matrix multiply

We first calculate P_{opt} by taking the derivative of Equation 1 and solving for P:

$$P_{opt} = (\frac{p}{m} * \frac{T_{comp}}{T_{comm}})^{\frac{1}{p+m}}$$

If we now calculate the ratio of the time spent on computation over the time spent on communication, or the Computation to Communication Ratio (CCR), and substitute P by P_{opt} we obtain:

$$CCR = \frac{T_{comm}}{T_{comp}} * P^{p+m} = \frac{p}{m} \tag{2}$$

We conclude that applications that follow this simple performance model will always have the same CCR when using the optimal number of nodes. This observation suggests a different method for adaptation. Applications can monitor how much time they spend on computation and communication, and depending on whether this ratio is too high or too low, they can increase or decrease the number of nodes they use, until the CCR reaches a value that is appropriate for this application. While this method is clearly less general than the model-based approach, it has the advantage that it is less sensitive to fine-grain measurements of low-level system parameters.

4 Matrix multiply - a simple example

In this section we apply the two adaptation methods outlined in the previous sections to a simple distributed matrix multiply. Adaptation is limited to estimating the optimal number of nodes at startup. We look at both an Ethernet and a router-based network.

4.1 Matrix multiply

The matrix-matrix multiply

$$C = A * B$$

in which A, B, and C are $N * N$ matrices, can be distributed over P nodes as follows. During a set up phase, we distributed blocks of rows of A across the P

processors (A_i on processor P_i), and we similarly distribute blocks of columns of B (B_i on processor P_i). The main computational phase consists of P steps in which each processor repeatedly multiplies a block from A with a block from B; in between steps the blocks from B are moved through the processors array in a ring fashion (Figure 3). At the end of the computational phase, each processor P_i will have calculated C_i, where C_i is the ith block of N/P rows in C. In a final phase, the result matrix C is returned to a central processor.

For the performance model we need the following additional parameters:

- b is the number of bits in a matrix element
- M is the number of bits in a submatrix, i.e. $M = b * \frac{N^2}{P}$
- T_{flop} is the time to execute a floating point operation (in seconds)
- BW is the bandwidth of the network (in bit/second), ignoring per-message overheads
- T_{fix} is the fixed cost of sending a message (in seconds)

For simplicity we assume that processors are homogeneous. We will consider two different environments. First, we will focus on an Ethernet topology: all workstations are connected to the same Ethernet. Since this is a bus-based topology, only one node can send at any point in time, so all nodes share the bandwidth BW. The second topology is a router or switch based topology where all nodes are connected by a private link to a single switch or router. All nodes can now potentially send (or receive) data at a rate BW simultaneously.

Given the above parameters we can easily derive the following model for each of the three phases of the distributed matrix multiply. We first focus on the Ethernet topology:

$$T_{initial} = P * (\frac{2 * M}{BW} + T_{fix}) \tag{3}$$

$$T_{computation} = \frac{N^3 * T_{flop}}{P} + (P-1) * (\frac{P * M}{BW} + T_{fix}) \tag{4}$$

$$T_{final} = P * (\frac{M}{BW} + T_{fix}) \tag{5}$$

During the initial (final) phase, there is only one sender (receiver), so $T_{initial}$ and T_{final} are the same for both networks. $T_{computation}$ is however Ethernet specific, since we assumed that each node can send data at a rate of $\frac{BW}{P}$. On a switch-based network, the transmit rate becomes BW and Equation 4 becomes:

$$T_{computation} = \frac{N^3 * T_{flop}}{P} + (P-1) * (\frac{M}{BW} + T_{fix}) \tag{6}$$

To validate the model, we used a testbed of 8 Alpha workstations model 255, connected by a dedicated 10 Mb Ethernet. The systems have 64 MByte of memory and run Dec Unix 4.0. We obtained the following system parameters:

- Using the matrix multiply itself as a benchmark, we measured $T_{flop} = 1$ microsecond.

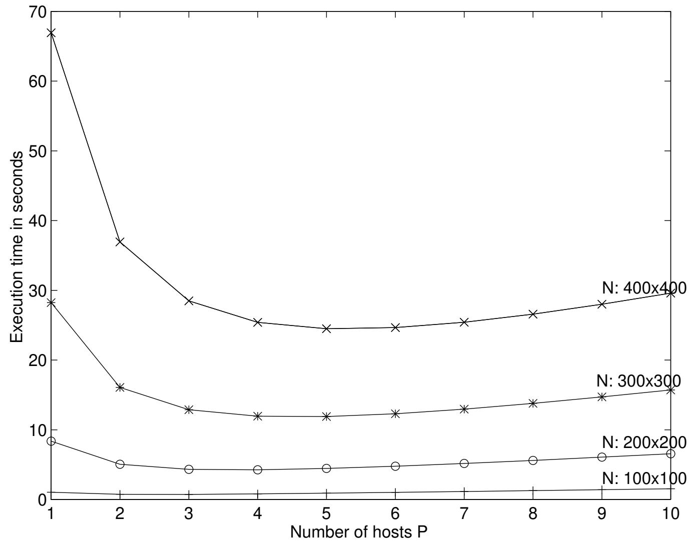

Fig. 4. Execution time as a function of the number of nodes

- To measure the network-specific system parameters, we measured the time to transfer blocks of two different sizes and then solve the equations for the two parameters. We found $BW = 8.8$ Mbs and $T_{fix} = 0.3$ millisecond.

The results in Figure 2 compare the results of the model for these parameters with measured execution times. We see that the model is quite accurate, although, as discussed in Section 2, the model is sensitive to the estimates for the system parameters.

4.2 Model-based estimation over Ethernet

If we take the derivative of the sum of Equations 3, 4, and 5, after replacing M by its value, and we solve for P, we get the following result (Ethernet):

$$\sqrt{\frac{N^3 * T_{flop} * BW}{b * N^2 + 3 * T_{fix} * BW}}$$

Given the shape of the curve representing the execution time as a function of the number of nodes (Figure 4), we set the number of nodes to the ceiling of this result, i.e. it is better to use too many nodes than too few,

$$P_{opt} = ceiling(\sqrt{\frac{N^3 * T_{flop} * BW}{b * N^2 + 3 * T_{fix} * BW}}) \tag{7}$$

We evaluate the effectiveness of this approach after we discuss ratio-based estimation.

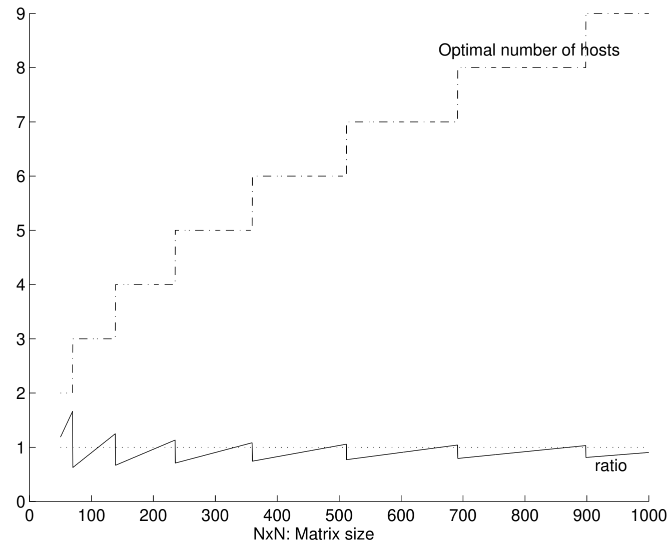

Fig. 5. Ratio of communication to computation time

4.3 Ratio-based estimation over Ethernet

Let us simplify the model described earlier in this section. In practice, the contribution of T_{fix} is negligible, so we will set it zero. When we also replace M by its value ($b * \frac{N^2}{P}$), we get the following simplified model (Ethernet):

$$T_{initial} = \frac{2 * b * N^2}{BW}$$

$$T_{computation} = \frac{N^3 * T_{flop}}{P} + (P-1) * (\frac{b * N^2}{BW})$$

$$T_{final} = \frac{b * N^2}{BW}$$

We observe that T_{inital} and T_{final} are independent of P, while $T_{computation}$ follows the model of Equation 1 in Section 3 with $m = p = 1$. This suggests that if the optimal number of processors is used, the communication to computation ratio should be (Equation 2):

$$CCR = \frac{p}{m} = 1$$

Figure 5 confirms this result. It shows the CCR, calculated using the detailed model (Equations 3, 4, and 5), with P set to P_{opt} (Equation 7). The CCR oscillates around 1; the oscillation is caused by the ceiling function.

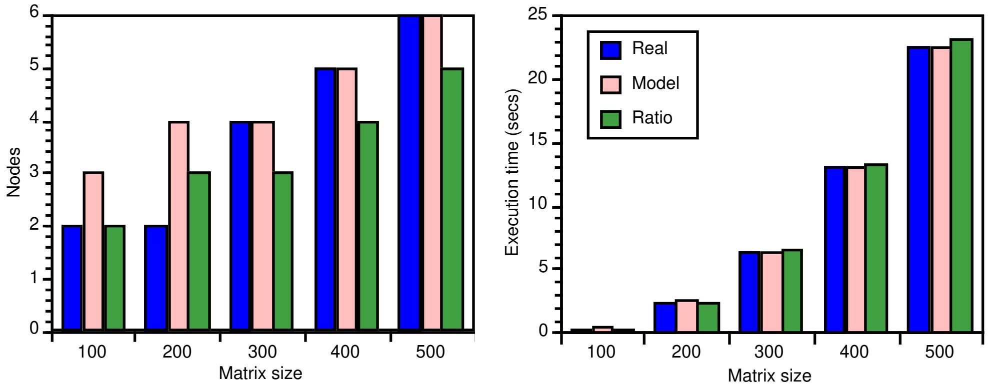

Fig. 6. Number of nodes (left) and execution time in seconds (right) with different estimation mechanisms - Ethernet

4.4 Performance over Ethernet

Figure 6(left) compares the "optimal" number of nodes selected by both methods for problems of different size with the "real" optimal number of nodes, determined experimentally using trial and error. Both estimation methods get quite close to the optimal; the maximum error is two nodes. For this particular example, model-based estimation tends to pick too many nodes for small problem sizes, while ratio-based estimation tends to pick too few nodes for larger problems. Figure 6(right) compares the execution times using the estimation methods with the minimal execution times. Again, the results are quite good. The error is within 5% for problem sizes over 100. The error is about 50% for model-based estimation on a problem size of 100, probably as a result of the very short execution times. One of the reasons for this good result is that the execution time is fairly insensitive to the number of nodes around the optimal number of nodes (Figure 4).

4.5 A router-based network

We replace the Ethernet interconnect by a routed topology: each of the 8 Alphastations is directly connected to a router using a point-point fast Ethernet (100 Mbs) link. The router consists of a Pentium Pro running NetBSD. We briefly look at both model-based and ratio-based estimation in this environment.

For model-based estimation we calculate the optimal number of nodes as we did in Section 4.2, but using Equations 3, 6, and 5. We find:

$$P_{opt} = ceiling(N * \sqrt{\frac{N * T_{flop} * BW - b}{3 * T_{fix} * BW}}) \tag{8}$$

For ratio-based estimation, we use the same method as in Section 4.3. Assuming the router is not a bottleneck, the simplified model for $T_{computation}$ for

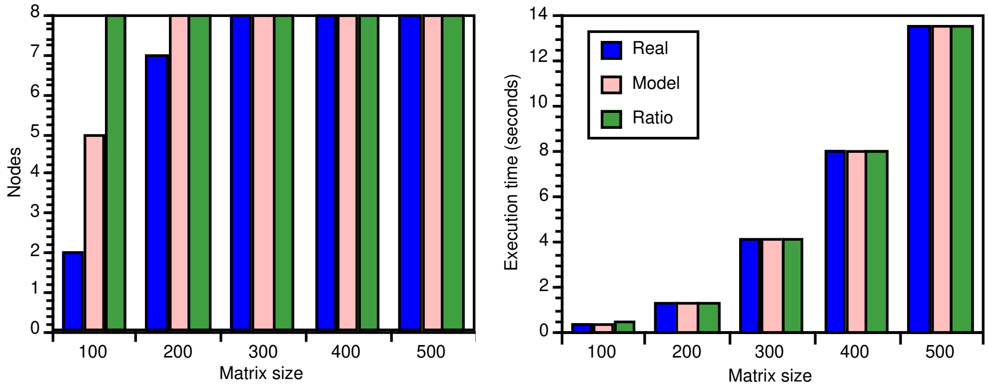

Fig. 7. Number of nodes (left) and execution time in seconds (right) with different estimation mechanisms - switched network

this topology (i.e. the model with $T_{fix} = 0$) is:

$$T_{computation} = \frac{N^3 * T_{flop}}{P} + (P - 1) * (\frac{b * N^2}{BW * P})$$

It roughly follows the model of Equation 1 with $p = 1$ and $m = 0$. This means that ratio-based would try to achieve a CCR equal to infiniti (Equation 2), by using as many nodes as possible. This result is not surprising, since in the simplified model, the communication cost does not increase with the number of nodes, while the computation cost decreases, so using more nodes is always better.

At a high level, ratio-based adaptation is giving us reasonable advice: in a switch-based environment, using many nodes is likely to be a good idea, although targeting an infinite number is of course unrealistic. The problem is that the simplified model breaks down if there are many nodes. Specifically, for very large numbers of nodes, T_{fix} will become significant since the per-byte costs will become negligible (very small messages). Moreover, for larger numbers of nodes the router may become a bottleneck, limiting aggregate throughput. Interesting enough, with both of these scenarios (router limiting aggregate throughput or T_{fix} become the dominating cost), the system ends up in a state where the model with $m = 1$ applies, so under those conditions, targeting a CCR of one may give good results.

We evaluated both model-based and ratio-based estimation on the router-based system, although the limited number of nodes in our system severely limits the evaluation. For example, we cannot demonstrate some of the limits of ratio-based adaptation that were described above. Moreover, the router does limit the throughput for larger numbers of nodes, so both of the estimation approaches will overestimate the number of nodes since they ignore that bottleneck. Figure 7 shows the results of the evaluation.

Ratio-based adaptation always uses the maximum number of nodes (eight), while model-based adaptation also uses eight nodes for all but the smallest problem. Figure 7 shows that this is the right thing to do. Clearly the usefulness of this evaluation is limited, since we cannot look at the more interesting processor range. Note that given the properties of the router, it may be more appropriate to model the network as an Ethernet with a high link speed (the aggregate throughput of the router is about 150 Mbs).

5 Related work

The problem of adaptation has received a fair bit of attention. In the networking area, transport protocols that do congestion avoidance (e.g. TCP/IP [8]) are an important example of adaptation to network conditions. In terms of application-specific adaptation to network conditions by *network-aware* applications, several groups have focused on continuous media traffic streams, e.g. network-aware voice [6, 2] and video [5, 14, 7, 9, 10] streaming applications.

There has been some work on non-multimedia network-aware applications, although the network-aware decision making is typically at a coarse level. One example is having storage-intensive applications (e.g. Web browsers) select the most appropriate server, based on the network conditions between the client and servers [3]. Another example is that of determining whether network conditions permit the offloading of tasks to other nodes. A lot of work on adaptive applications is done in the context of wireless networks, since network resources tend to be both scarce and variable in this context [10].

Distributed computing applications running over (shared) networks tend to be much more complicated and they have received less attention. Some results are presented in [11, 1]. A related problem is that of mapping applications on distributed-memory system [12]. This problem is somewhat easier, since these systems have static, well-understood performance characteristics. Mapping is typically based on a performance model of the application that includes both computation and communication costs, and mapping often requires profiling to determine the execution cost of application modules.

6 Conclusion

In this paper we studied two approaches to making a distributed computing application network-aware. We specifically looked at the problem of selecting the optimal number of nodes for the application. The first approach is based on a performance model that captures the performance of the application as a function of system and application parameters. The second method tries to determine the optimal number of nodes by balancing the time spent on computation and communication. Using a very simple distributed computation, we show that both methods can approximate the optimal number of nodes fairly well. The example also shows the limitations of both methods. The model-based method requires an accurate performance model, which may be difficult to construct for more complex applications, and it requires accurate measurements of system parameters, which may be application-specific. The ratio-based approach seems to be less sensitive to details of the application and system, but the particular method presented in this paper is restricted to a limited class of applications. Clearly more work is needed to find algorithms that are both robust and general.

References

1. J. Bolliger and Thomas Gross. A framework-based approach to the development of network-aware applications. Submitted for publication, 1997.
2. Jean-Chrysostome Bolot and Andres Vega-Garcia. Control mechanisms for packet audio in the internet. In *IEEE INFOCOM'96*, volume 1, pages 232–239, San Francisco, CA, March 1996. IEEE.
3. Robert Carter and Mark Crovella. Server selection using dynamic path characterization. In *IEEE INFOCOM'97*, volume 3, pages 8C–4, Kobe, Japan, April 1997. IEEE.
4. Tony DeWitt, Thomas Gross, Bruce Lowekamp, Nancy Miller, Peter Steenkiste, and Jaspal Subhlok. ReMoS: A Resource Monitoring System for Network Aware Applications. Technical Report CMU-CS-97-194, Carnegie Mellon University, December 1997.
5. R. Frederick. Network video (nv), 1993. Software available via ftp://ftp.parc.xerox.com/net-research.
6. V. Jacobson and S. McCanne. Visual audio tool (vat), 1993. Software available via ftp://ftp.ee.lbl.gov/conferencing/vat.
7. V. Jacobson and S. McCanne. Vic, 1995. Software available via ftp://ftp.ee.lbl.gov/conferencing/vic.
8. Van Jacobson. Congestion Avoidance and Control. In *Proceedings of the SIGCOMM '88 Symposium on Communications Architectures and Protocols*, pages 314–329. ACM, August 1988.
9. H. Kanakia, P. Mishra, and A. Reibman. An adaptive congestion control scheme for real time packet video transport. *IEEE/ACM Transactions on Networking*, 3(6):671–682, December 1995.
10. Brian Noble, M. Satyanarayanan, Dushyanth Narayanan, James Tilton, Jason Flinn, and Kevin Walker. Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth Symposium on Operating System Principles*, pages 276–287, October 1997.
11. Bruce Siegell and Peter Steenkiste. Automatic selection of load balancing parameters using compile-time and run-time information. *Concurrency - Practice and Experience*, 9(3):275–317, 1996.
12. Jaspal Subhlok and Gary Vondran. Optimal latency-throughput tradeoffs for data parallel pipelines. In *Proc. SPAA'96*. ACM, June 1996.
13. Hongsuda Tangmunarunkit. Middleware for network-aware distributed computing applications. Undergraduate thesis, School of Computer Science, Carnegie Mellon University, May 1997.
14. Hideyuki Tokuda, Yoshito Tobe, Stephen Chou, and Jose Moura. Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network. In *Proceedings of the SIGCOMM '92 Symposium on Communications Architectures and Protocols*, pages 88–98, Baltimore, August 1992. ACM.