Adaptation Models for Network-Aware Distributed Computations

Peter Steenkiste
School of Computer Science
Carnegie Mellon University
e-mail: prs@cs.cmu.edu

Abstract

Network-aware applications actively adapt to the level of service they receive from the network. This allows the application to execute well over a diverse set of networks and under a wide range of network conditions. However, network diversity and dynamic network conditions make the development of network-aware applications a difficult task, since the developer has to be an expert in both the application domain and networking. In this paper we look at a number of network-aware applications and identify three adaptation strategies that have proven to be effective. These strategies can be viewed as adapation models that capture the essential structure of the adaptation process. Similar to the use of programming models in parallel and distributed computing, adaptation models can be used to guide the development of other network-area applications and they can also form the basis for programming support, e.g. middleware, that supports the development of network-aware applications. We describe these three adaptation models, compare their features and applicability, and briefly discuss how these models impact the design of middleware that supports network-aware applications.

1 Introduction

Clusters of PCs or workstations have become a widely used architecture for compute-intensive applications because of their attractive price-performance ratio. Clusters consist of a collection of high-performance PCs or workstations connected by a commodity network. Clusters can be dedicated, i.e. they sit in a machine room and are used exclusively for high-performance computing, or they can be distributed and shared, i.e. the machines are scattered in offices, labs and machine rooms and are used for a diverse set of tasks. While both types of clusters may have a similar hardware characteristics, distributed clusters are a more challenging environment, because the higher degree of resource sharing results in a continuously changing environment. This is especially true for the network, which is by its very nature a shared resource, and we often find the highest diversity (ATM, switched Ethernet, Myrinet, ..). This presents a challenge to applications since the level of service they can expect from the network is unpredictable. It can easily change by as much as three orders of magnitude from one invocation to the next, and by a smaller factor during a particular invocation. This means that communication operations can take anywhere from milliseconds to seconds depending on the environment and current network conditions.

For some applications, e.g. *telnet*, the only possible response to changing network conditions is to speed up or slow down proportionally with the level of network service [6], and they can use protocols such as TCP [11] to do this. However, more complex applications often have a richer set of options for how to adapt to changing network conditions. A simple example is a video streaming application. Changes in available network bandwidth should not result in simply sending the same video data at slower or faster rate. Instead, the video source can respond by changing the video frame size or the video frame rate. Similarly, in a congested network, a distributed computation may want to reduce the number of nodes it uses instead of seeing its communication time go up significantly. We call applications that *actively* adapt to the level of service they receive from the network network-aware applications.

Developing network-aware applications is difficult because the developer must have expertise in both the application domain and networking. Libraries and middleware could potentially help applications adapt to network conditions, but in the network-aware applications that exist today, the adaptation process is mostly ad hoc and highly application-specific, complicating the design of broadly applicable support. As a first step towards developing such middleware, we have to better understand the fundamental adaptation processes. In this paper we argue that, while the details of the adaptation algorithms may differ, existing adaptive applications mostly use one of a small number of "good" adaptation strategies. The main contribution of this paper is that we identify three "adaptation models" that characterize widely applicable adaptation strategies.

The rest of this paper is organized as follows. We first give some examples of network-aware applications and describe

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

the general structure of such applications (Section 2). In Sections 3 through 5 we present three different adaptation models. We then discuss the relative advantages and disadvantages of the models, and their impact on the design of middleware (Section 6). We conclude with related work and a summary.

2 Network-aware applications

We describe the general structure of network-aware applications using some examples.

2.1 Examples

Applications can adapt to network conditions in a variety of ways. A first possibility is that applications change how much data they send. Data volume can for example be reduced by reducing the frame size of a video stream [10], or by dropping the pictures from a document (plain text instead of postscript, Web pages with or without images). Other ways of modifying how much data is sent is to apply (lossless) compression; this approach trades compute resources for communication resources. In some cases domain specific adaptation can be used. For example, when rendering an image on a remote node, the source can transmit the rendered image or rendering commands, depending on how much network bandwidth and computational resources are available.

The options for adaptation for distribution computing applications are typically much more restricted than the above examples may suggest. The reason is that distributed computations typically want reliable data transfers so it is not possible to drop data while options such as compression will often be too expensive. However, adaptation is often still possible. For example, applications can change what nodes or how many nodes they use. Examples are deciding to perform certain tasks locally or remotely [13], or determining the optimal number of nodes to use for a distributed computation [17, 18].

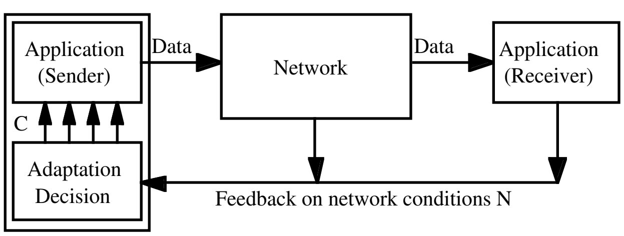


Figure 1. The adaptation process

2.2 The adaptation process

The goal of adaptation is to optimize application performance, given a certain level of service from the network. This process is illustrated for a simple point-to-point application in Figure 1:

- A change in the network is observed at a measurement point, e.g. a switch in the network or an application task running on the destination. The measurement point must measure the disturbance with a certain accuracy, translate it in information that is of use to the application, and send the information to the source or sources.
- The application makes a decision on adaptation, and waits until it can adapt. Depending on the application this may be short, e.g. every frame of a video transfer, or long, e.g. the application has to reach some synchronization point.
- The application adapts. Depending on the application this can happen quickly, e.g. change a parameter in a device, or it can be expensive or can be a slow process, e.g. state has to be moved between nodes.

Figure 1 shows two different feedback mechanisms. With *implicit* feedback, the application monitors end-to-end performance, and uses that to derive information about the status of the network, e.g. dropped packets can be viewed as a sign of congestion by both protocols (TCP [11]) and applications (vic [10]). With *explicit* feedback, the network provides explicit information about the status of the network to endpoints. While explicit feedback is used in some communication protocols [1, 5], it is not very common because of concerns that it may add too much complexity to the network. However, there is a growing recognition that explicit feedback can simplify the development of network-aware applications [7, 12].

The effectiveness of adaptation will depend on several features of the feedback loop in Figure 1. A first factor is the delay in the feedback loop: the shorter the delay, the better the application will be able to track changes in the network. The second factor is the accuracy of the feedback information. Clearly inaccurate information can degrade performance by having the application adapt needlessly, or by having the application select an incorrect operating point. A final factor is less obvious but quite significant in practice. There often is a cost associated with adaptation, which means that improving efficiency by adapting may not always result in a performance gain. To determine whether adaptation will pay off, applications have to compare the cost of adaptation with the potential benefits. The cost of adaptation and the performance gain can typically be estimated based on information about the current and "optimal" state. The performance benefit is typically proportional to time, so the condition for adaptation becomes

$$AdaptationCost < Benefit = Gain \times Duration$$

Unfortunately, the duration of the benefit is typically unknown since it depends on future conditions, e.g. network conditions could change immediately after the adaptation operation, so in a best effort environment there is no guarantee that adaptation will pay off.

2.3 Adaptation models

Whether adaptation is effective depends strongly on the what adaptation Decisions the application makes, i.e. the nature of the application decision module in Figure 1. While the

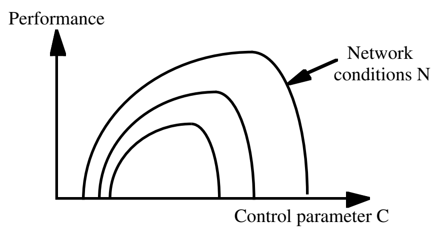


Figure 2. Adaptation decision process

specific adaptation algorithms may be application-specific, we argue that only a small number of mechanisms are used by today's applications. We will call these "adaptation models", similar to the programming models used in parallel and distributed computing. Identifying these models is a first step towards providing middleware support for adaptation. We identify three such models in the next three sections.

3 Model-based adaptation

In the first adaptation model, the application has a model of its performance as a function of the various parameters characterizing the runtime environment, e.g. network latency, ... Given information on the runtime environment, the application uses the model to select the settings for the control parameters that will give the best performance. We will call this *model-based adaptation*. We elaborate on the methodology and provide some examples.

3.1 The adaptation decision

In the simplest case, the adaptation decision module has to select a single control parameter (e.g. increasing or decreasing frame rate), and there is a simple relationship both between the control parameter and application performance (e.g. higher rate is better) and between the control parameter and the required network service (e.g. higher frame rates require more bandwidth) (Figure 2). In general, multiple control parameters have to be selected, and the relationship between application performance, control parameters, and network service is complex.

The adaptation decision process can be captured in a few simple formulas. The application performance P is a function p of the network conditions N and the values of a set of control parameters C:

$$P = p(N, C)$$

The challenge is to find the values for C that maximize P, given the network conditions N.

In reality, neither the network conditions or the dependency between P, N, and C are known and understood completely accurately. The following formulation of the adaptation problem captures this uncertainty: given a measured set of conditions $N_{measured}$ and a performance model p_{model} of the application, determine the values C_{opt} for the control parameters that maximize the the performance, i.e. $p_{model}(N_{measured}, C_{opt})$ is maximized. C_{opt} is a function of $N_{measured}$:

$$C_{opt} = c_{model}(N_{measured})$$

where c_{model} is based on p_{model}.

In model-based adaptation, the function p_{model} is an explicit formula that is derived by the application designer. c_{model} is calculated by setting the derivative of p_{model} with respect to C equal to zero, and by solving for C. This formula for C_{opt} is then used by the adaptation decision module at runtime to calculate the "optimal" values for the control parameters, based on the measured network conditions. Information about network conditions can be collected by the application by running a built in set of benchmarks, or from the network by querying a special interface [7, 13].

In reality using C_{opt} will of course not result in optimal performance in any formal sense. The systems involved are too complex to be modeled completely accurately, so p_{model} is only a rough approximation of reality p_{real}. Similarly, noise, delay, and errors will cause $N_{measured}$ to be only an approximation of the real network conditions N_{real}. Ideally, these effects will have little impact on performance, i.e.

$$P = p_{real}(N_{real}, c_{model}(N_{measured}))$$

is high and is not very sensitive to errors in $N_{measured}$ and c_{model} (i.e. p_{model}).

The above equation also shows that adaptation is in general application specific since both c and p are application specific. However, some components of the implementation are application independent, so it may be feasible to develop adaptation libraries that can be customized by the application. Collecting network status information or a callback mechanism to notify the application of significant changes in the network are examples of support that could be provided [7].

3.2 Examples

Most examples of model-based adaptation are in applications that make a simple decision based on a very simple application model. Examples include selecting the best file server or Web server for replicated data (highest bandwidth wins) [4], or a decision on whether a computation should be off-loaded (available bandwidth has to be above some threshold) [14].

A more complex application using model-based adaptation is described in [17]. It is a Successive Over Relaxation (SOR) application that has been distributed through pipelining (Figure 3). The input matrix is distributed in block-column fashion across the nodes, with a one-column overlap between adjacent blocks. The computation is pipelined: each node processes G rows before it passes on new values for the shared column to its right neighbor and receives values for its left neighbor. Given a data set and a number of nodes, the performance is primarily determined by the grain size G. G de-

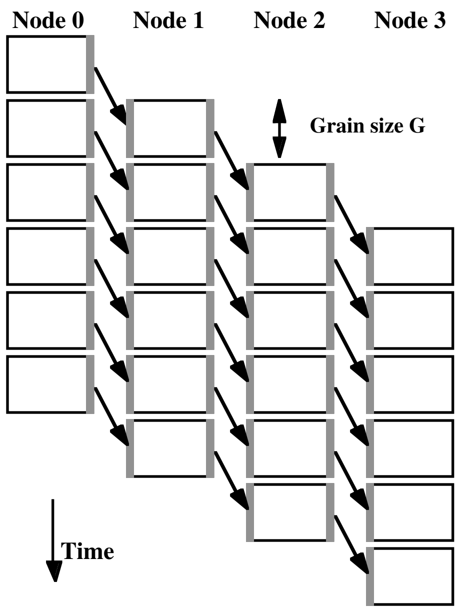


Figure 3. Pipelined SOR

termines how much computation is performed between communication steps and its optimal value depends on the performance of the network: as network performance increases, smaller grain sizes become more effective since they reduce the pipeline startup and drain cost.

The application was made network-aware by building a model that represents the execution time as a function of the grain size, the floating point performance of the nodes, and the performance of the network, modeled as a per-messages cost and a per-byte cost (see [17] for details). From this model one can derive the grain size that results in the minimal execution time, as described above. The optimal grain size is a function of the network and host parameters, which are estimated by having the application execute small built-in benchmarks at start up. Figure 4 illustrates how adaptation works. It shows how the efficiency (inverse of the execution time) changes with grain size. The graph includes curves representing measured efficiencies on a workstation cluster, estimated efficiencies based on the application model and measurements of communication parameters, and an upperbound on efficiency (communication cost is zero). The vertical line represents the grain size that is selected by model-based adaptation. We observe that it achieves close to the actual best efficiency. We found similar results for networks with different performance characteristics [17, 16].

Adaptation in SOR has two interesting features that seem to be quite common in distributed computations. First, adaptation is expensive, both because the nodes have to synchronize and because they have to reshuffle a lot of application data. This overhead will limit the frequency of adaptation. Second, adaptation tradeoffs are asymmetric. For the SOR application, it is better go underestimate network performance than to overestimate it.

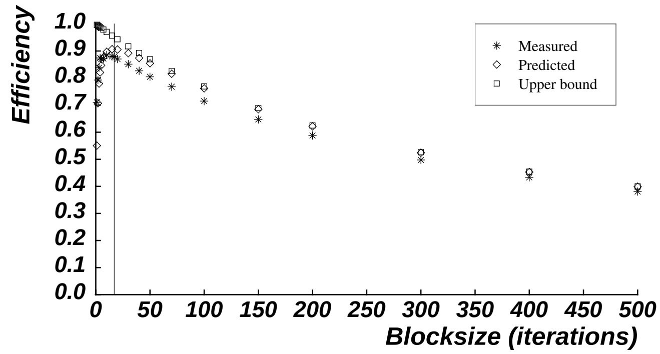


Figure 4. Model-based adaptation in SOR

3.3 Discussion

Model-based has the advantage that it can potentially quickly zoom in on the right control parameter values. It can be used at startup or to periodically reevaluate control parameters during execution. While we have only limited experience, a disadvantage of this approach seems to be its robustness: the performance is very sensitive to both the accuracy of the network status information and the correctness of the model. Building an accurate model is clearly difficult for complex applications, although one can argue that some form of model will be needed for any method of adaptation. Getting reliable estimates for the system parameters also turned out to be difficult. One of the reasons is that appropriate estimates for system parameters are often application specific, e.g. different application have widely different floating point performance, and network throughput may depend on features such as message size and protocols used. This application deals with this by using segments of the application to estimate the system parameters, but this complicates application development.

4 Performance-based adaptation

In a second model, the application continuously monitors its own Performance and it changes its control parameters based on these observations. The control parameters are typically updated incrementally, because there is not enough information available to explicitly calculate the parameters. While model-based adaptation is based on explicit network information, performance-based adaptation collects and uses information about the network implicitly.

4.1 Examples

Congestion control in the widely used TCP transport protocol can be viewed as an example of performance-based adaptation. It optimizes performance by maximizing the rate at which it can send data reliably. If the receiver reports lost data, the sender reduces its rate, on the assumption that the data loss was due to network congestion, and reducing the rate will thus result in (more) reliable transmission.

The prototypical example of application-level network-aware adaptation is adaptive video transmission [19, 8, 9].

Similar to TCP, the video streaming applications try to maximize their throughput, while trying to eliminate or minimize data loss. Again, the receiver provides feedback to the sender about packet loss, but in this case the sender will reduce its transmission rate by reducing the frame rate, frame size, or the number of bits per pixel.

4.2 Discussion

The advantage of performance-based adaptation is its simplicity, at least for applications that have a simple definition of "performance" that depends in a straightforward manner on network performance. We are not aware of any complex applications that fit this model. This approach does not need a detailed model or accurate measurements of system parameters, and as a result, it is likely to be more robust. A disadvantage is that this method is purely reactive, i.e. the application adapts after performance has degraded. Moreover, changes will typically be incremental because the accuracy of the information is much lower; this can be expensive if there is a cost associated with adaptation. Finally, performance-based adaptation typically supports only "one-way" adaptation: the application can observe that it is overloading the network (observes poor performance) but it cannot easily detect whether the network can support a higher load. A solution is to periodically probe the network by increasing the load, using the effect on performance to determine whether the network can handle it. Probing can be difficult to manage and introduces extra overhead.

5 Feature-based adaptation

In a third model, the application monitors some feature of the application and uses that to adapt. This is possible if the feature has a known "good value" that correspond to a good application state, and deviations can be translated into changes of the control parameters. We will call this *feature-based adaptation*. This can be viewed as a generalization of the previous model, in the sense that performance is an application feature. However, by picking the feature carefully, it might be possible to achieve more desirable adaptation behavior. Improvements are possible in two areas. First, it might be possible to adapt before there has been a significant loss in performance, i.e. adaptation may be preventive instead of reactive. Second, adaptation is possibly symmetric, i.e. adaptation can increase and reduce the load on the network without having to periodically probe for additional resources.

5.1 Example

An example of feature-based adaptation in transport protocols is TCP Vegas [3]. TCP Vegas continuously monitors the roundtrip time, and compares it with a roundtrip time estimate that is based on a model of the roundtrip time as a function of throughput. By comparing the measured and estimated roundtrip time, the sender can determine how its rate compares with the available rate on the bottleneck link, so it can adapt without necessarily having to wait for packet loss.

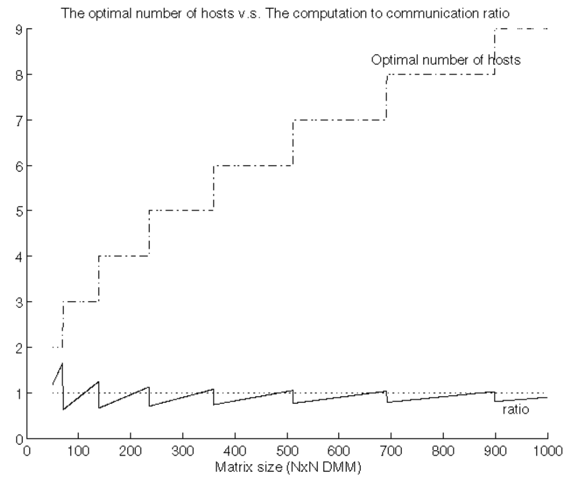


Figure 5. Optimal communication to computation ratio in MM

An example of application-level feature-based adaptation is described in [18]. The problem addressed is to determine the optimal number of nodes for a distributed matrix multiply. A first solution is to use model-based adaptation. While model-based adaptation works, it is very sensitive to the measurements of the system parameters, as we already observed for the SOR application. An alternative is to use feature-based adaptation, using the communication to computation ratio as the feature being monitored. As described in [18], for a certain class of parallel application, this ratio is a known constant when the optimal number of nodes is used, independent from the network conditions. For the distributed matrix multiply, the communication to computation ratio is one when the optimal number of nodes is used. This is illustrated in Figure 5. So by monitoring this ratio, and increasing (decreasing) the number of nodes if the ratio is higher (lower), one can stay at a good operating point. Figure 6 compares the performance of both model-based and feature-based adaptation with the experimentally-determined best performance. We observe that both mechanisms do a good job estimating the optimal number of nodes, and the execution times are also close to optimal.

5.2 Discussion

Feature-based adaptation shares many of the characteristics of performance-based adaptation. One difference is that it does not require probing. It is however not clear how well this approach generalizes: it is difficult to identify an application feature that reliably indicates how well the application's current use of the network fits the network conditions. Note also that neither performance-based or feature-based adaptation work at all for adaptation at startup, since the methods are based on feedback from a running application.

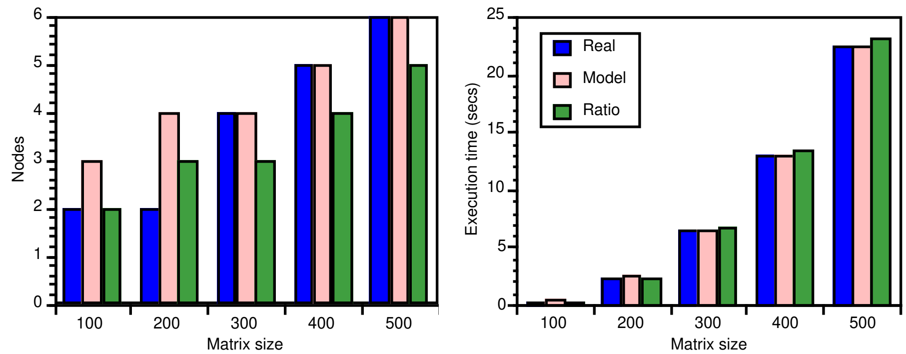
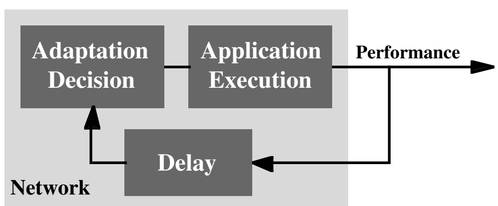
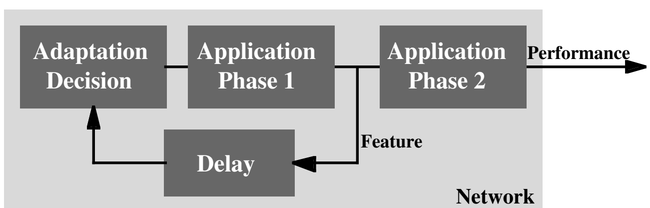


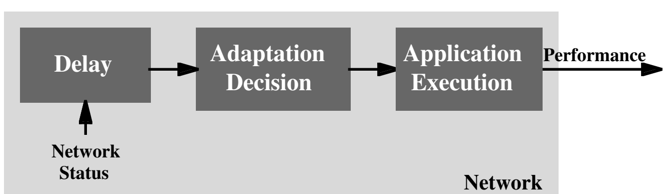
Figure 6. Performance for the network-aware MM: number of nodes (left) and execution time (right)



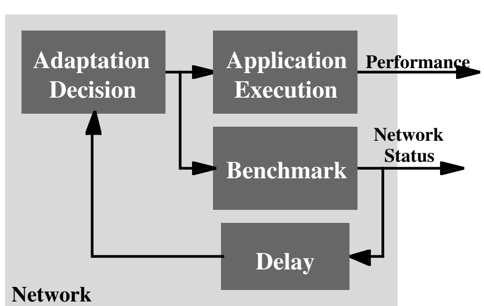
(a) Performance-based adaptation



(b) Feature-based adaptation



(c) Model-based adaptation based on explicit network information



(d) Model-based adaptation based on application benchmark

Figure 7. Adaptation represented as control loop

6 Applicability of the adaptation models

We can clarify the differences between the three models by presenting them as simple control loops, as is shown in Figure 7. Performance-based adaptation has the simplest loop (Figure 7(a)): the receiver provides feedback on the observed performance and the adaptation module in the sender uses this information to change the setting of the control parameters. This is only possible if the appropriate control parameters can be derived directly from the observed performance.

Feature-based adaptation has a very similar structure (Figure 7(b)), except that part of the application is outside of the control loop. The main impact of this change is on robustness: if the relationship between the feature that drives adaptation and performance is not correctly understood, the system will adapt to the wrong operating point. However, selecting an appropriate feature may simplify adaptation compared with performance-based adaptation.

Figure 7(c) shows model-based adaptation based on information provided by the network. We note that there is no closed-loop feedback: application-level adaptation is done open loop. This means that both the adaptation decision process (based on an application model) and the measurements will have to be fairly accurate, otherwise the adaptation process can easily go astray. Figure 7(d) shows model-based adaptation based on information collected by the application. This is still an open loop process: the benchmark typically does not characterize end-to-end application performance but only tries to characterize the network. However, the benchmark can be customized by the application, increasing the level of confidence that appropriate data is being collected.

Figure 7(a), (b), and (d) can be viewed as three points on a spectrum. The variable in the spectrum is what fraction of the application is "covered" by the feedback loop.

Based on these differences in the models, we can make some high-level observations about the applicability of the models. Table 1 lists some important features of each of the models. Robustness characterizes what fraction of the application is covered by the closed loop feedback. Timeliness characterizes how quickly the application can learn about changes in network conditions. In general, the simpler the feedback loop, the quicker it can respond. This means that benchmarks are likely to provide feedback more quickly than complete applications, although benchmarking is not likely to be a continuous activity. Feedback based on monitoring

	Robustness adaptation	Timeliness information	Nature of adaptation	Locality information
Performance	high	low	incremental	local
Feature	medium	medium	incremental	local
Model	low	high	jump	broader

Table 1. Features of different adaptation models

may give the quickest response. The last two features affect the type of information that is collected. Performance-based and feature-based adaptation provide "relative" information that in general only allows incremental adaptation. Model-based adaptation provides absolute information that in principle should allow adaptation to an optimal state. The information obtained as part of performance-based and feature-based adaptation is also limited in scope (i.e. whatever network information is observable through the performance or feature) and restricted to the part of the network used by the application. In contrast, model-based adaptation can use a broader set of information.

We conclude that all the models have both desirable and undesirable features, and which model is most appropriate will depend on the application and the environment. The features in Table 1 provides some guidance. Note that application may want to use a combination of models. Model-based adaptation can be used to select an appropriate operating region from a potentially very large space of operating options that cover both network topology and application set up; this can be done at startup and periodically throughout execution. More frequent incremental runtime adaptation can be based on the performance or feature-based models.

7 Middleware support for network-aware applications

We discuss a number of ways in which middleware can support network-aware applications.

7.1 Network information

The simplest way in which middleware can help network-aware applications is to provide information about the network conditions. This is especially true for applications that use model-based adaptation since they rely critically on precise low-level network information. If the application can simply query the network to find out about network properties such as application-level throughput and latency, then it does not have to do its own benchmarking. This should simplify code development significantly, and will also reduce the level of network expertise needed by the application developer since network benchmarking is a difficult task. Applications will often be interested in different types of information, e.g. delay versus bandwidth or behavior on different time scales, so network should provide access to a rich set of information.

Another feature of adaptation that should be considered when developing middleware is that adaptation tradeoffs are often asymmetric, so the application would like to know the accuracy of the information. For example if the network information is highly variable, the SOR application would lower its estimate of the available network bandwidth (more conservative), while it might be more optimistic if the bandwidth estimate is known to be accurate.

The Remos interface [7, 12] developed at CMU is a first attempt at define the interface to such a middleware layer. It has been implemented and is being evaluated.

7.2 Application structure

Making an application network-aware requires profound changes in the application structure. An interesting approach to simplify this task [2] is to use frameworks to capture the control flow of the adaptation process. Application can fill in the application specific tasks. Figure 7 shows graphically some possible structures for simple point-to-point applications. Clearly, a richer set of options for adaptation exists for multi-party applications with more than two end-points. The middleware could for example coordinate the movement of data (state) that has to happen as part of the adaptation, or can provide the necessary synchronization between the nodes involved in adaptation. However, work is needed in this area.

7.3 Service selection

An important issue for any network-aware application is the predictability of network conditions. If network conditions are very unpredictable, effective adaptation will be difficult, if not impossible (Section 4). This suggests that we should build networks that are predictable. The networking community is addressing this problem by implementing networks with multiple classes of service [6]. Besides best effort service, which is what today's networks support, networks can for example support guaranteed service [15], which offers highly predictable service, albeit at a higher cost. Another example is controlled load service [20], which maintains a minimum level of service for applications by limiting the number of users in the network. So far, there is little experience in using these service classes.

8 Conclusion

Network-aware applications actively adapt to the level of service they receive from the network. In this paper we take a first step towards providing structured support for network-aware applications by identifying three adaptation models that capture the essential properties of effective adaptation. Model-based adaptation uses a performance model of the application and measurements of the network conditions to determine the optimal operating point for the application. Performance-based adaptation is based on monitoring application performance and adapting if the performance drops. Feature-based adaptation is a generalization of performance-based adaptation: adaptation is based on an application feature other than performance. Using examples, we discussed the advantages and disadvantages of the models and their applicability.

References

[1] ATM Forum Traffic Management Specification Version 4.0, October 1995. ATM Forum/95-0013R8.

[2] J. Bolliger and Thomas Gross. A Framework-based Approach to the Development of Network-aware Applications. *IEEE Transactions on Software Engineering*, 24(5):376–390, May 1998.

[3] L. S. Brakmo, S. W. O'Mally, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of ACM SIGCOMM'94*, London, UK, August 1994.

[4] 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.

[5] Prashant Chandra, Allan Fisher, Corey Kosak, and Peter Steenkiste. Experimental evaluation of atm flow control schemes. In *IEEE INFOCOM'97*, pages 1326–1334, Kobe, Japan, April 1996. IEEE.

[6] Dave Clark, S. Shenker, and L. Zhang. Supporting realtime applications in an integrated services packet network: Architecture and mechanisms. In *Proceedings of the SIGCOMM '92 Symposium on Communications Architectures and Protocols*, pages 14–26, Baltimore, August 1992. ACM.

[7] 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.

[8] R. Frederick. Network video (nv), 1993. Software available via ftp://ftp.parc.xerox.com/net-research.

[9] J. Inouye, S. Cen, C. Pu, and J. Walpole. System support for mobile multimedia applications. In *Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 143–154, St. Louis, May 1997.

[10] V. Jacobson and S. McCanne. Vic, 1995. Software available via ftp://ftp.ee.lbl.gov/conferencing/vic.

[11] Van Jacobson. Congestion Avoidance and Control. In *Proceedings of the SIGCOMM '88 Symposium on Communications Architectures and Protocols*, pages 314–329. ACM, August 1988.

[12] Bruce Lowekamp, Nancy Miller, Dean Sutherland, Thomas Gross, Peter Steenkiste, and Jaspal Subhlok. A Resource Query Interface for Network-aware applications. In *7th IEEE Symposium on High-Performance Distributed Computing*. IEEE, July 1997.

[13] Brian Noble, M. Price, and M. Satyanarayanan. A Programming Interface for Application-Aware Adaptation in Mobile Computing. *Computing Systems*, 8, Fall 1995.

[14] 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.

[15] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service, September 1997. IETF RFC 2212.

[16] Bruce Siegell. *Automatic Generation of Parallel Programs with Dynamic Load Balancing for a Network of Workstations*. PhD thesis, Department of Computer and Electrical Engineering, Carnegie Mellon University, 1995. Also appeared as technical report CMU-CS-95-168.

[17] 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.

[18] Hongsuda Tangmunarunkit and Peter Steenkiste. Network-aware distributed computing: A case study. In *Second Workshop on Runtime Systems for Parallel Programming (RTSPP)*, Orlando, March 1998. IEEE. Springer-Verlag.

[19] 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.

[20] J. Wroclawski. Specification of the Controlled-Load Network Element Service, September 1997. IETF RFC 2211.