CHAPTER 9: CONCLUSIONS AND FUTURE WORK

9-1. Numbers of Arrangements and Configurations

We began by asking whether the numbers of arrangements and configurations for given numbers of processes and processing nodes are very large. If not, then it may be possible to handle placement decisions on an ad hoc basis. In Chapter 3 we showed that the numbers grow very large very quickly as the numbers of processes and processing nodes increase. With just four processes and up to four processing nodes there are 15 possible configurations in five arrangements. With 16 processes on up to 16 processing nodes, moderate numbers of each, there are $1.05 \cdot 10^{10}$ configurations in 231 arrangements. With 64 processes on up to 64 processing nodes there are $1.72 \cdot 10^{65}$ configurations in $1.74 \cdot 10^{6}$ arrangements. Clearly these numbers are too large to handle on any ad hoc basis. In Chapter 3 we found formulas for computing the numbers of arrangements and configurations for given numbers of processes and processing nodes, and for computing the number of configurations in a specific arrangement.

9-2. Experimental Design

Our experimental design uses simulation of artificially generated traces. The traces cover all possible combinations of relative amounts of CPU computation, I/O actions, and communication activity. For some investigations we used the *full grid*, which covers all possible combinations, for others we used a *zoomed-in grid*, which focuses on the lowest CPU, highest communication part of the space.

Our measure of performance is the response time of the distributed computation, that is, the elapsed wall clock time from the moment all processes begin execution until the moment the last process finishes.

9-3. Differences in Performance

If the differences between the performance of one configuration and another are small, then it is not necessary to carefully select which configuration to use. One configuration will work about as well as another, and the placement decision can be made randomly, or arbitrarily, or on some basis other than response time performance.

The differences in response time between configurations are substantial. Over the full grid, the mean differences between the response times of the fastest configuration and the slowest configuration are about 30% with two processes, about 98% with three processes, and about 152% with four processes. The sizes of the differences vary considerably from run to run; from almost nothing up to almost 50% with two processes, almost 200% with three processes, and almost 300% with four processes. The mean differences are clearly significant, and the largest differences are very large. Even the mean differences between the fastest configuration and the second-fastest average 30-35%. It would seem to be worthwhile to explore this problem, and to design, implement, and install a placement procedure. The time required at program execution time, before the program can begin executing, to make a placement decision can be more than made up by the improvement in response time due to making a good choice of configuration.

The APART configuration, in which each process is executed on a separate processing node, is almost always fastest, and the TOGETHER configuration, in which all processes are executed together in the same processing node, is nearly always slowest. The response times of other configurations fall between these extremes.

The response times tend to grow as the proportion of CPU execution actions in the traces grows, since the mean CPU service request is longer than the mean time to service an I/O or a communication action. The differences between the response times of different configurations also tend to grow as the proportion of CPU grows.

The response times of the configurations fall into clusters, with differences of 30-50 percentage points between clusters and differences of only a few percentage points between configurations within clusters. The clusters correspond to groups of configurations with the same *collocation number*, the largest number of processes executing on any one processing node.

9-4. CPU Service Determines Best Configuration

In Chapter 6 we carried out a statistical analysis to identify the variables which would best allow us to choose the best configuration. We found that the mean CPU service time requested in execute actions is the single best variable, and that it has most of the predictive power of larger groups of variables. We used the regression formula with that single variable to predict the difference between the response times of the two configurations possible with two processes. The relative errors in the predictions were very large; nonetheless, predicting which of the two configurations would be faster based on the predicted difference was successful in about 80% of the runs.

In Chapter 7 (Section 7-3) we tried two different approaches to selecting the fastest of the three-process 2:1:0 ‘‘loner’’ configurations: matching, and attractive and repulsive forces. Applying the matching approach over the full grid, we found that the highest success rate, about 83% successes, was achieved by selecting as the loner process the process with the largest CPU demand, using data from the APART simulations and the ‘‘eql’’ approach. Applying the attractive and repulsive forces approach over the full grid, the highest success rate, about 77% successes, was achieved using information about the CPU demands of the processes. We performed the attractive and repulsive forces analysis using data from TOGETHER simulations; we believe that with data from the APART simulations the success rate would be approximately the same as the matching success rate, about 83% successes. The fact that, in both approaches, using the CPU demand data led to the highest success rates confirms the results of the statistical analysis: CPU demand is the most important factor in selecting a configuration, and leads to 80-85% success rates over the full grid. (If real distributed programs are distributed uniformly over the full grid, we can expect these techniques to select the best configuration about 80-85% of the time. Real programs are probably concentrated in one region of the grid, and the actual success rates will reflect the success rates in that part of the grid.)

In the zoomed-in grid, which zooms in on the most low-CPU, high-communication part of the grid, communication is as important, or more important, than CPU demand in reaching high success rates with the matching and attractive/repulsive forces approaches.

In Chapter 8 (Section 8-7) we tried to select the best configuration out of each arrangement according to which configuration had the smallest difference between the CPU demands on the different processors; we called this the CR-even approach. The success rates were about 64% for the four-process 2:1:1:0 ‘‘triples,’’ about 57% for the 2:2:0:0 ‘‘pairs,’’ and about 89% for the 3:1:0:0 ‘‘loners.’’ Although these success rates are lower than we had hoped, they are several times better than random selection. Furthermore, there is evidence that in many or most of the unsuccessful cases, the configuration chosen as fastest was in fact the second or third fastest, which would make it on average only around 5% slower than the fastest configuration. So this is another way in which using CPU demand information leads to reasonably good placement decisions.

In the traces we generated, all processes have similar, though not identical, characteristics. Thus, the differences in CPU demand from one process to another are small. Despite this, placement techniques based on the differences between the CPU demands of different processes are reasonably successful. In

some real programs, the differences between the CPU demands of different processes may be larger, in which case these placement techniques should be even more successful.

We analyzed the ratio of synchronization delay, the time processes spend waiting for other processes to send them messages, to communication delay, the amount of time processes spend waiting for messages to travel through the communication network. (In Section 6-7.4 we called this ratio the synchronization proportionality factor, or SPF.) Over the full grid, the mean ratio is about 153, that is, processes spend about 153 times longer waiting for other processes to send them messages than they wait for the messages to arrive. Even in the zoomed-in grid, which is extremely communication-intensive, the mean ratio is about four. These values are remarkably high, especially considering that our traces are generated in such a way that processes are unlikely to do a receive unless another process has sent or soon will send a message to them. This may explain why CPU demand is more important than communication in selecting a configuration.

Spreading processes out onto different processing nodes parallelizes, and thus speeds up, computation. Synchronization delay is a matter of how much computation a process must do before it reaches the point of sending its next message; the faster the process can complete that computation, the smaller the synchronization delay, so spreading processes out reduces the synchronization delay. The communication delay between processes on different processing nodes is much larger than that between processes on the same processing node, so spreading processes out increases the communication delay. Grouping processes together onto the same processing nodes reduces the communication delay between them, but at the same time it reduces parallelism and slows down the computation of the processes. The large synchronization-to-communication delay ratios mean that processes spend relatively large amounts of time waiting for other processes to complete computations, and relatively little waiting for messages to be transmitted. Thus, speeding up computation is much more important in improving response time than is speeding up communication. Since spreading processes out speeds up computation at the expense of slowing down communication, spreading out processes as much as possible is desirable.

Since response time does not depend very strongly on network speed, our results should apply to metropolitan-area networks (MANs) and wide-area networks (WANs), even if they are much slower than the networks we model, and even if the speed of message transmission is not uniform between all source/destination pairs, as is our model.

9-5. Maximum Number of Processes Collocated on a Processing Node

We call the maximum number of processes collocated on any processor the *collocation number*. Arrangements with the same collocation number have similar response times, on average over the full grid, even when the number of processes and the number of processing nodes are different. The smaller the maximum number of processes on any node, the faster the mean response time. This does not mean that the collocation number is sufficient information to predict response time; programs in different part of the full grid have very different response times. It does mean that, for a fixed number of processes, using more processing nodes and reducing the collocation number will lead to a faster response time; in other words, on average it is good to spread the processes out as much as possible.

All processes in a trace are similar, so the collocation number is a crude measure of the maximum CPU demand on a processor. We have seen that CPU demand is the most important determinant of response time within an experiment set. The similarity of the average response times of arrangements from different experiment sets which have the same collocation number further supports the idea that CPU demand determines response time.

Mean response times do increase with increasing collocation numbers, but the increase is much less than linear. For example, the increase in mean response time between 1-collocated and 2-collocated arrangements is about 53%, not 100%. The increase between 1-collocated and 4-collocated arrangements

is about 190%, not 300%. This shows that when processes share a processing node, they are able to overlap use of the facilities; for example, one process might be using the CPU while another is performing an I/O. The larger the number of processes sharing the node, the more they are able to overlap in this manner.

Although CPU demand is the strongest determinant of response time, communication does have an influence. The mean response time over the full grid of the 2:0 configuration is faster than that of the 2:1:0 configurations, though both types are 2-collocated. We believe the slowdown is due to additional synchronization delay, time spent waiting for messages from the third process. The 2:1:1:0 triples and 2:2:0:0 pairs are all 2-collocated, but the mean response time of the triples is lower. We believe this is because the two processes which are spread out in the triple configurations send their messages earlier than when the processes are collocated, so other processes experience less synchronization delay. These results contradict Indurkhya, Stone, and Xi-Cheng [ISX86], and agree with Nicol's mathematical analysis, though not with the simulations Nicol reports [Nic89] (see Section 2-3.2).

9-6. The Always-APART Placement Policy

The APART configuration, the configuration which spreads processes out as much as possible, is the fastest configuration in almost all cases. The exceptions are programs with extremely low proportions of CPU actions and extremely high proportions of communication actions, making up no more than about 1.4% of the combinations in the full grid. We believe that real, scientific programs are characterized by high CPU use and more communication than I/O; such programs fall into a different part of the grid than the region in which APART is not always best. Even in the few cases in which APART is not the fastest configuration, it is generally only a few percent slower than the fastest configuration. Thus, a very simple, yet very effective placement policy when there are enough processing nodes available is to always use the APART configuration; we call this the always-APART policy. This policy would be very easy to implement, thus reducing the possibility of programming errors, and it would be the fastest possible policy at program execution time since no decision needs to be made. Since the policy does not use any information, there is no need to collect and store information, nor to periodically update the information, nor to worry about the effects of inaccurate or out-of-date information. It is hard to imagine that any other placement procedure could be better or cheaper. So, if there are enough processing nodes available that each process can be executed on its own processing node, simply using the APART configuration is an efficient and effective approach.

9-7. Spreading Out the CPU Demand

In all the situations we have studied, spreading processes out as much as possible nearly always leads to the fastest response time. When there are enough processing nodes available, APART is nearly always the fastest configuration by far. In the 3x2 situation (three processes to be executed, with two processing nodes available), a 2:1 loner configuration is faster than 3:0 TOGETHER. In the 4x3 situation a 2:1:1 triple configuration is faster than a 2:2:0 pair, a 3:1:0 loner, or 4:0:0 TOGETHER. The best placement procedures we have found, always-APART when there are enough processing nodes, matching on most CPU demand, CPU demand repulsion, and CR-even, all spread out the CPU demand as much and as evenly as possible.

We believe that spreading out the CPU demand as much and as evenly as possible is the best approach to making placement decisions for any numbers of processes and processing nodes. This means using as many processing nodes as are available (up to the number of processes). In a NxN situation, in which there are enough processing nodes, an APART arrangement of the form 1:1:...:1:1 should be fastest. In a Nx(N − 1) situation an arrangement of the form 2:1:1:...:1:0 should be fastest. Specifically, based on the results of the four-process 2:1:1 triples and 2:2:0 pairs, we expect that arrangements of the form

2:1:1:...:1:0 will be faster than arrangements of the form 2:...:2:0:...:0.

9-8. Marginal Advantage of Adding One Processor

We have computed the marginal advantages, the reductions in response time obtained by adding one processor. The mean-to-mean margins compare the mean response time of all configurations using some number of processors to the mean response time of all configurations using one more processor. The mean-to-mean margins are about 20% to 30%. The largest margin corresponds to going from one processor to two, and is about twice as large as the other margins.

The best-to-best margins, comparing the best response time using some number of processors to the best response time using one more processor, are the ones which should be used to compare the performance with one number of processors to that with another number of processors if it were possible to identify the best configurations using those numbers of processors. The best-to-best margins range from about 11% to about 40%. Disappointingly, the margins do not have a clear trend, they go from larger to smaller to intermediate as the number of processors increases.

The best-to-worst margins compare the best response time using some number of processors to the worst response time using one more processor. This is the minimum ‘‘guaranteed’’ benefit of adding a processor. When going from one to two processors, or three to four, the best-to-worst margins are about 20%. Between two and three processors, however, the margin is about -1.60%, an increase in the response time rather than a decrease; this reflects the overlapping response time ranges of the triple- and pair-type configurations.

9-9. Future Work

These results suggest several avenues for continuing and extending our research.

We would like to see whether our results apply when there are larger numbers of processes and processing nodes. In our work so far, we have simulated each trace in all possible configurations. With the combinatorial explosion in the number of configurations, it will not be possible to continue in this manner. We will have to apply our results so far to select an ‘‘interesting’’ subset of configurations which will be sufficient to confirm our previous results and, hopefully, allow us to draw new conclusions.

We would like to explore the sensitivity of our results to changes in the system characteristics, especially to changes in the speed of communication. We have seen that the processes of distributed programs spend far more time waiting for other processes to send them messages (synchronization delay) than they do waiting for the messages to reach them (communication delay); on average, for two-process programs over the full grid, the synchronization delay is 153 times larger than the communication delay. Our results should therefore apply to distributed systems in which the communication speed is much different, or in which the speed of communication is not uniform between pairs of communicating processes.

We have modelled very uniform programs and systems: all process in a trace have similar characteristics; processes are equally likely to send a message to any other process in the program; all processing nodes are identical; the speed of communication is the same between any pair of nodes; and there is no outside load. We would like to remove some of these constraints and see what effect that has on performance, and on how placement decisions should be made. The GENACTIONS trace generator can generate traces in which clusters of processes communicate heavily between them, but communicate only infrequently with processes outside the cluster. The Carnival simulator can simulate processing nodes in which the CPUs execute at different speeds, and it can simulate outside CPU, I/O, and communication load.

We believe that (1) processes with different service demand characteristics, (2) heterogeneous processing nodes with different service rates, and (3) outside load on node facilities can be handled in similar

ways in making placement decisions. Replacing one process running alone on a node with a process which has twice the CPU demand should have the same effect on response time as replacing the processor with one half as fast. Adding a competing outside process with the same CPU demand should have a similar effect. We would like to explore this possibility.

We would also like to explore performance and placement in systems in which the components differ in more than their service rates. For example, processing nodes with different architectures, and even processing nodes which are themselves multiprocessors. Do our results apply to shared-disk systems, such as workstations with a common file server? Do they apply to systems made up of specialized components, such as not-so-powerful personal computers backed up by very powerful compute servers and file servers?

All our work has been done with artificially generated traces; we have explained the advantages of this. It would be interesting to obtain traces from some real distributed programs, to see where in the grid their characteristics fall, and to see whether they behave as our results predict.