CHAPTER 7: THREE PROCESSES

7-1. Introduction

In this chapter we discuss distributed programs which consist of three processes. When there are three processes to be placed and three processors available — a *3x3 situation* (read ‘‘three by three’’) — the processes may be placed on one, two, or three processors. Not all available processors must be used; one of the goals of this research is to discover when it is best to use one, two, or three processors. When all three processors are used, it is called a *3@3 usage* (read ‘‘three on three’’). In a 3x3 situation, the placement procedure must choose a 3@1, 3@2, or 3@3 usage. Similarly, when there are three processes to be placed and two available processors — a *3x2 situation* — the placement procedure must choose whether to use one processor or two. If only one processor is available — a *3x1 situation* — there are no placement decisions to make, as all three processes must share the one available processor in a 3@1 usage. Since there are no placement decisions to make, no analysis of the 3x1 situation is necessary.

In the remainder of this introductory section we describe the arrangements and configurations possible in 3x3 and 3x2, and the experiments used to gather data about them. In Section 7-2 we present the analyses done on the experimental results to explore the 3x3 situation, while in Section 7-3 we present the 3x2 analyses. Section 7-4 contains analyses of three particular regions of the full grid.

7-1.1. 3x3 Arrangements and Configurations

7-1.1.1. 3x3 Arrangements

An *arrangement* describes the number of processes assigned to each processor, but does not map specific processes to processors. In the 3x3 situation there are three possible arrangements of processes on processors (see Figure 7-1):

Situations			Usage	Arrangement	Configuration	Label
3x3	—	—	3@3	1:1:1	0 / 1 / 2	APART
"	3x2	—	3@2	2:1:0	12 / 0 / -	LONER0
"	"	—	"	"	02 / 1 / -	LONER1
"	"	—	"	"	01 / 2 / -	LONER2
"	"	3x1	3@1	3:0:0	012 / - / -	TOGETHER

Figure 7-1:
Arrangements and Configurations
of Three Processes

(1) the all-apart arrangement, 1:1:1, in which one process runs on each of three processors; (2) the all-together arrangement, 3:0:0, with all three processes running on one processor, with the other two processors unused. These two arrangements are analogous to the 2x2 all-apart and all-together arrangements, respectively. The third 3x3 arrangement has no 2x2 analog: (3) the *loner* arrangement, 2:1:0 (or 2:1), with two processes on one processor, one process on a second processor, and the third processor unused. This is called the loner arrangement because one process is alone on a processor.

7-1.1.2. 3x3 Configurations

A *configuration* is a mapping of specific processes to processors. There are one or more configurations in each arrangement. There is only one possible configuration in the 1:1:1 arrangement: 0/1/2 (process 0 on one processor, process 1 on a second processor, and process 2 on the third processor). All processors are identical, so it does not matter which process is assigned to which processor; by convention, we write 0/1/2 rather than 1/0/2 or 2/1/0. Since each of the three processes runs on a separate processor, apart from the others, this is called the 3x3 *APART* configuration.

The 3:0:0 arrangement also has only one possible configuration: 012/-/- (processes 0, 1, and 2 on one processor, none on the other processors). Since all the processes are together on one processor, this is called the 3x3 *TOGETHER* configuration.

The 2:1:0 arrangement is a bit more interesting. Although we are assuming that all processors are identical, each process is different, so it does matter which two processes are placed together and which one is alone. There are three possible configurations: 12/0, 02/1, and 01/2. We call these *loner configurations,* because, in each, one process is alone on a processor. The three configurations are called LONER0, LONER1, and LONER2, respectively, depending on which process is the loner; see Figure 7-1.

7-1.2. 3x2 Arrangements and Configurations

The arrangements and configurations possible in 3x2 are a subset of those possible in 3x3, the subset which uses two or fewer processors. The 1:1:1 arrangement and the 0/1/2 APART configuration are not possible in 3x2, but all the other arrangements and configurations possible in 3x3 are also possible in 3x2.

All five 3x3 configurations were simulated for each trace, and all five were used in analyzing the 3x3 situation. When analyzing the 3x2 situation it is not necessary to run further simulations; the 3x2 analyses are performed using only a subset of the 3x3 configurations.

7-1.3. Experiment Sets

7-1.3.1. Introduction

The 3x3 and 3x2 work is based on the same grids used in the 2x2 work; the grids specify the characteristics of processes but not the number of processes in each trace. Experiment set U is based on the full grid, and experiment set V is based on the zoomed-in grid.

In 2x2 there are only two processes, so each process has only one possible communication partner. With three processes, each process has two possible communication partners, so the question arises, to which destination process or processes will each process send messages, or what will be the ratios of the numbers of messages sent to each destination? Although GENACTIONS has the capability of generating clusters of processes that communicate mostly amongst themselves (see Section 5-3.7), we have chosen for simplicity to generate traces with uniform communication. Each sending process is equally likely to choose as destination any of the other processes.

7-1.3.2. Experiment Set U

Experiment set U is based on the full grid. Experiment set N, discussed in the preceding chapter, is also based on the full grid, but where the traces in set N consisted of two processes, each trace in set U consists of three processes. The full grid has 26 cells; in set U they are labelled **ua**, **ub**, and so on, through **uz**, similar to the labelling of set N; see Figure 6-1. As in set N, there are two repetitions per cell, for a total of 52 runs. Each trace is simulated five times, once in each configuration, making a total of 260 simulations.

7-1.3.3. Experiment Set V

Experiment set V is based on the zoomed-in grid, as is two-process set R. The zoomed-in grid has 36 cells; in set V they are labelled **vaa**, **vab**, and so on, through **vbj**; see Figure 7-12. As in set R, there are four repetitions per cell, for a total of 144 runs in set V. Each trace is simulated once in each of the five configurations, making a total of 720 simulations.

7-2. Three Processes with Three Processors (3x3)

7-2.1. Introduction

Our main goal is to find effective algorithms for the placement procedure to use in selecting the configuration to be used. As in the 2x2 situation, we decided to run a full factorial experiment, so that every configuration of each trace would be simulated. In the following sections we discuss which configurations are fastest in which circumstances, the sizes of the differences between response times of different configurations, and some candidate placement policies. Full-grid set U is discussed in Section 7-2.2, and zoomed-in-grid set V is discussed in Section 7-2.3.

7-2.2. Experiment Set U as 3x3

7-2.2.1. Introduction

In this section we discuss a number of analyses of the outcomes of the set U simulations. We note first which configuration has the fastest response time in each run (Section 7-2.2.2), followed by a presentation of some of the response times (Section 7-2.2.3). In Sections 7-2.2.4 and 7-2.2.5 we analyze the differences between the response times of the configurations. In Sections 7-2.2.6 and 7-2.2.7 we discuss two candidate placement policies: always-APART and dyadic placement. Finally, in section 2.2.8, we consider the conclusions which can be drawn from these analyses of set U.

7-2.2.2. Winning Configurations

In every run except those in **ua** (the upper left cell), APART is the configuration with the fastest response time, the *winning configuration.* Even in the **ua** runs, the response times of the APART configurations are only about 4% slower than the response times of the fastest configurations; the exact figures are shown in Figure 7-2.

Run	Winner	Difference Between Winning Config. and APART	
		Absolute Difference	Relative Percentage Difference
ua-1	LONER1	57 ms	4.10%
ua-2	LONER2	62 ms	4.50%

Figure 7-2:
Differences Between the Response Times of
the Winning Configuration and APART

The relative percentage difference is computed based on the best response time; the formula used is $[(t_{best} - t_{APART}) / t_{best}] \cdot 100$.

In every run except **ua-1** and **ua-2**, the three loner configurations are second, third, and fourth best, but the order varies from run to run (of course, the numbering of the processes is arbitrary, so this means nothing). In every run, without exception, TOGETHER is the worst configuration.

7-2.2.3. Response Times

There are 260 response times for set U (26 cells, 2 traces per cell, 5 configurations per trace). To provide a general impression of the response times in different parts of the grid, Figure 7-3 shows the response times of all five configurations in each of the corner cells in set U.

	IO-COMM				
CPU	IO 0	COMM 100	...	IO 100	COMM 0
0	**ua**			**ue**	
	1,390	1,372		11,162	11,244
	1,400	1,383		19,563	19,553
	1,406	1,392	...	19,793	19,975
	1,447	1,434		20,465	20,439
	1,509	1,502		29,852	29,940
...	...		...	...	
80	**uu**			**uy**	
	40,125	45,264		36,104	35,622
	58,867	65,121		66,041	64,328
	62,926	66,560	...	66,862	65,084
	67,478	67,304		67,526	66,996
	93,132	93,792		99,983	97,891
100	**uz**				
	41,690	41,200			
	78,552	80,583			
	80,783	81,127			
	81,150	81,857			
	120,243	121,784			

Figure 7-3:
Response Times of the Corner Cells in Set U (in ms)

Each column within a cell contains the response times for one trace, ordered from fastest time on top to slowest on the bottom.

In most of the runs shown, the fastest response time is considerably faster than the second fastest, the second, third, and fourth response times have similar response times, and the slowest response time is much slower than the next fastest one. This pattern is less clear in runs **ua-1** and **ua-2**, but it is quite clear in the other runs shown in Figure 7-3. This pattern will be confirmed and quantified in Section 7-2.2.4.2.

Figure 7-4 shows summary statistics for the response times of each of the five configurations, of the three loner-type configurations combined, of all configurations combined, and of the one winning (fastest) configuration in each run.

Max. Number of Procs. on Node	Config. Types	Number of Runs	Response Times (ms)			
			Mean	Std. Dev.	Minimum	Maximum
1	APART	52	26,166	12,891	1,434	50,826
2	LONER0	52	38,961	21,918	1,392	81,127
2	LONER1	52	38,251	21,472	1,383	81,150
2	LONER2	52	37,695	20,760	1,372	81,857
3	TOGETHER	52	54,713	31,889	1,502	121,784
2	all loners	156	38,302	21,395	1,372	81,857
—	all configs.	260	39,157	24,369	1,372	121,784
—	winning config.	52	26,136	12,895	1,372	50,826

Figure 7-4:
Response Times of Configurations in Set U

The mean APART response time is almost as small as the mean winning response time, reflecting the fact that APART is usually the winning configuration. We will discuss the response times further in Section 7-2.2.5.

7-2.2.4. Relative Differences

7-2.2.4.1. Introduction

In this section we discuss the differences between the response times of the five configurations simulated for each run. The sizes of the differences indicate how much improvement in response time can be obtained by selecting a good configuration, or, conversely, how much of a penalty will be suffered for not selecting a good configuration. The sizes of the differences may also indicate groups of configurations with similar performance.

Both absolute differences in milliseconds and relative percentage differences are presented. Relative percentage differences are all computed relative to the winning response time of the run; for example, the relative percentage difference between the second best and third best configurations is $[(t_{secondbest} - t_{thirdbest}) / t_{best}] \cdot 100$.

7-2.2.4.2. Step Differences

In each run, we computed the differences between the best response time and the second best, between the second best and the third best, and so on; we call these *step differences*.

The mean step difference is 7,137 ms, with standard deviation 8,852 ms; the smallest is 4.94 ms, and the largest is 39,926 ms. The mean relative percentage step difference is 24.51%, with standard deviation 24.95%; the smallest is 0.02%, and the largest is 96.91%. The mean step difference is a substantial fraction of the best response time, indicating that the selection of a good configuration is important.

Figure 7-5 shows the mean step differences over set U for each step separately.

	Absolute Difference (ms)			
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	10,526	9,015	9.91	39,383
2nd-Best – 3rd-Best	1,482	1,716	4.94	9,076
3rd-Best – 4th-Best	1,879	2,611	41.22	13,836
4th-Best – Worst	14,664	9,730	61.23	39,926

	Relative Percentage Diff.			
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	35.72	22.05	0.71	95.59
2nd-Best – 3rd-Best	5.06	4.28	0.02	19.82
3rd-Best – 4th-Best	6.10	5.49	0.46	27.22
4th-Best – Worst	51.17	20.06	4.40	96.91

Figure 7-5:
Mean Step Differences for each Step in Set U

Figure 7-5 indicates that there are three clusters of configurations. The best configuration is about 36% better, that is, considerably better than the next best; the second, third, and fourth best configurations all have similar performance; and the worst configuration is about 51% worse, that is, much worse than the fourth best configuration. In almost every run in set U, APART is the best configuration, the three loner configurations are second, third, and fourth best, and TOGETHER is worst. Figure 7-5 indicates, therefore, that the three loner configurations all have similar performance, and that APART, TOGETHER, and the cluster of loners have significantly different performances. We can also observe this clustering in the mean response times of the configurations, shown in Figure 7-4.

7-2.2.4.3. Best/Order Differences

Best/order differences are the differences between the response times of the best configuration and the second-best configuration, between the best configuration and the third-best configuration, and so on. We have computed all the best/order differences for each run. Figure 7-6 gives the mean best/order differences over all 52 runs in set U.

	Absolute Difference (ms)			
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	10,526	9,015	9.91	39,383
Best – 3rd-Best	12,007	9,453	15.78	39,926
Best – 4th-Best	13,886	10,510	57.00	40,657
Best – Worst	28,550	20,089	118.23	80,583

	Relative Percentage Diff.			
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	35.72	22.05	0.71	95.59
Best – 3rd-Best	40.78	21.88	1.14	96.91
Best – 4th-Best	46.88	22.51	4.10	98.68
Best – Worst	98.05	41.98	8.50	195.59

Figure 7-6:
Mean Best/Order Differences in Set U

If the differences between the best response times and the worst response times were small, it would not matter which configuration was used. Figure 7-6 shows that the differences between best and worst response times are quite large. On average, the worst response time is about twice as long as the best (98.05%). Even the smallest best/worst difference is close to 10% (8.50%); in one case the worst time is almost triple the best time (195.59%). Unless doing so is very costly (takes a lot of time), it is probably worthwhile to try to avoid the worst configuration.

Is it necessary to identify the very best configuration? The mean best/second-best difference is about 36%. This is a substantial fraction of the response time, so it is probably worthwhile for the placement procedure to identify the very best configuration, but it depends on how long the placement procedure takes. At worst, the second-best response time is almost twice the fastest; it is very probably worth the cost of selecting the very best configuration to avoid the possibility of doubling the response time. For small response times a substantial slowdown is wasteful, and possibly irritating to the user; for large jobs requiring hours of CPU time, doubling or tripling the response time may be disastrous in terms of human-scale time delay or in terms of computation costs, or both.

7-2.2.4.4. The Effect of Dissimilar Processes on Relative Differences

In traces generated by GENACTIONS, all processes in a trace have similar characteristics. Would the relative differences be larger or smaller if the processes were less similar?

Dissimilar processes would make little difference in the APART configuration. Each process has its own processing node; it can use whichever of its facilities it wishes whenever it wishes.

In TOGETHER, the processes compete for use of shared resources in one processing node. There is probably considerable overlapping of facility use: when one process is using the disk another can use the CPU, and so on. When more than one process makes heavy use of one facility, there is less possibility of overlap. If the processes were less similar, there would be more possibility of overlap. If the processes are similar but well balanced, it is not clear whether making them less similar would increase overlap or not.

The traces in full-grid set U span the spectrum from very unbalanced facility use (processes use one facility heavily) to well-balanced, so it is not clear what the overall effect of generating dissimilar processes would be on the response time of TOGETHER. Assuming that unbalanced traces would run faster, and that well-balanced ones would have response times similar to the ones in our experiments, the net effect would be a small to moderate speedup in the TOGETHER response time. Since the APART time would not change, the TOGETHER/APART difference would shrink somewhat, and hence so would the best/worst difference.

The effect of dissimilar processes on the response times of the loner configurations would depend greatly on the characteristics of each particular process and on the particular configurations. We have seen that, when the processes are all similar, it does not make much difference which one is the loner. If the processes were dissimilar, at least one combination of two of the processes should allow for considerable overlap, resulting in a faster response time. Since the APART time would not change, the difference between APART and the best of the loners should be smaller, and hence so would the best/second-best difference.

If real programs consist of processes which are less similar to each other than the processes in these experiments, the differences may be smaller than the results reported here. However, it is likely that many distributed programs consist of several similar worker processes (also called slave or child processes), at least in some phases. It is such programs that our traces model, and to which the results discussed above apply.

7-2.2.5. Response Time and Relative Difference as a Function of Number of Processes

The mean response times of 2x2 set N are shown in Figure 6-6; for 3x3 set U they are shown in Figure 7-4. Notice that the 2x2 APART times are similar to the 3x3 APART times; in both configurations, one process runs on each processing node. Notice also that the 2x2 TOGETHER times are similar to the 3x3 loner times; in both configurations, two processes run on one of the processing nodes. It is not surprising that the largest number of processes running on a processor determines the performance of a configuration. We will call the largest number of processes running on any processor in a given configuration the *collocation number* of the configuration. The APART configurations are 1-collocated, the 2x2 TOGETHER and 3x3 loner configurations are 2-collocated.

In both groups of configurations, the mean response time of the three-process configuration is larger; the increase is 5.38% for the 1-collocated APART configurations and 1.72% for the 2-collocated configurations (using the mean of all 3x3 loner response times to compute the latter). The increase is probably due to additional time spent waiting for remote messages to arrive.

In 2x2 set N, the average magnitude relative percentage difference between APART and TOGETHER is about 30%, and the values range from about 6% to about 50%; see Figure 6-13. In 3x3 set U, the mean relative percentage differences between APART and the loner-type configurations range from about 36% to about 47%; the actual values range from less than 1% to about 97%; see Figure 7-6.† In both groups, the difference is between a 1-collocated configuration and a 2-collocated configuration. Generalizing, then, we can say that on average across the full grid, changing from any 1-collocated configuration to any 2-collocated configuration will increase or decrease the response time by somewhere in the range of 30-50%.‡ Generalizing based on only two data points is risky, especially when the data points are quite different; we believe the difference between the two-process and three-process cases is due to different communication characteristics. Also, we have been discussing the mean differences; the ranges of differences in individual runs are very large.

In both the cases examined here, the number of processing nodes used has changed by one. It is not clear whether similar results would be obtained if the number of processors changed by a larger number, for example, from a 1-collocated eight-process all-apart arrangement (1:1:1:1:1:1:1:1) to a 2-collocated eight process pairs arrangement (2:2:2:2).

According to Figures 6-5 and 6-6, moving between a 2-collocated configuration and a 3-collocated configuration changes the response time by about 51-62%, and moving between a 1-collocated configuration and a 3-collocated configuration changes the response time by about 98%, but these figures are based only on the three-process results.

We believe that APART will nearly always be the best configuration in any NxN situation. We also believe that the next best configuration will nearly always be in an arrangement of the form 2:1:...:1:0, and that the best configuration in any NxM situation where $M = N - 1$ will also nearly always be in such an arrangement. Such an arrangement has two processes collocated and the rest of the processes each on their own processor, so we can expect the response time to be about 30-50% slower than the APART time. Other arrangements with no more than two processes on a processor, for example 2:2:1:...:1:0:0, should have similar response times. Furthermore, arrangements in which three processes share a processor will have response times about 100% slower than the APART time. These conjectures are based only on the two-process and three-process results and may not hold for larger numbers of processes and processing nodes. These conclusions may be very useful in deciding whether it is worth procuring additional processing nodes.

---

† The figures for set U are the best/order differences between the fastest configuration and the second-, third-, and fourth-fastest configurations. In every run except **ua-1** and **ua-2** APART is fastest and the loners are second-, third-, and fourth-fastest. Because of the **ua** runs, the best/order differences are only approximately the APART/loner differences.

‡ Part of the difference between the 2x2 and 3x3 results is due to the fact that the 2x2 relative percentage differences are computed relative to the response time of TOGETHER, which is usually the largest response time, while the 3x3 relative percentage differences are computed relative to the fastest, that is, smallest, response time for each run.

What happens as the number of processes collocated on the same processor continues to increase? Will the increase in response time level off or will it continue to increase? We cannot tell from the two- and three-process results, but we speculate that the size of the slow-down will continue to increase, but it will increase more and more slowly as the number of processes continues to increase. We will examine this question further in Chapter 8.

7-2.2.6. The Always-APART Policy in Set U

In Section 6-6 we described the always-APART policy, a very simple placement policy in which the configuration chosen is always APART. The simplicity of the policy makes it easy to implement, and fast to execute at placement time. Its drawback is that APART is not always the best configuration.

In set U, APART is the fastest configuration in all runs except **ua-1** and **ua-2**. The characteristics of cell **ua** are probably very unrealistic; it is highly unlikely that any real distributed program would do nothing but send and receive messages, doing almost no computation. Even in cell **ua** the response time of APART is only about 4% slower than the fastest response time. Thus, the always-APART policy is highly successful in set U; it nearly always hits, and even when it misses the penalty is quite small.

7-2.2.7. Dyadic Placement in Set U

In Chapter 6 we found that we could make fairly good predictions of the best two-process configuration, APART or TOGETHER, with the PRED1 technique, using the variable *smextime*, which is the mean of the CPU requests made in the execute actions of the two processes (see Section 6-8). In 3x3 we have three processes, so this technique can not be used directly, but we can apply it indirectly.

Consider the three processes as three pairs of processes, or *dyads*, as shown in Figure 7-7.

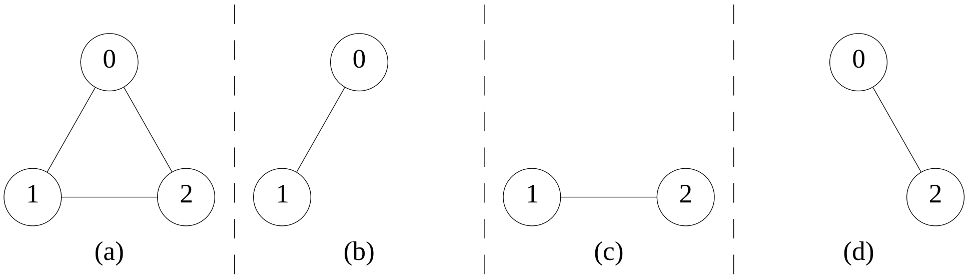

Figure 7-7:
Three Processes as Three Dyads:
(a) Three Processes, (b)-(d) Dyads
Processes numbered 0, 1, and 2
form the dyads 0-1, 1-2, and 2-0

We can apply the PRED1 technique to each dyad separately, yielding a result for each dyad, specifically, whether the processes in the dyad should be placed together on a processor or whether they should be placed on different processors. Combining the three dyad results may yield a conclusion about how to place all three processes. For example, if all three dyads should be broken apart, then the three processes should be placed in the three-process APART configuration; if all three dyads should be together, then the processes should all be placed together. It is also possible for a combination of dyad aparts and togethers to yield a 3x3 loner placement, as shown in Figure 7-8.

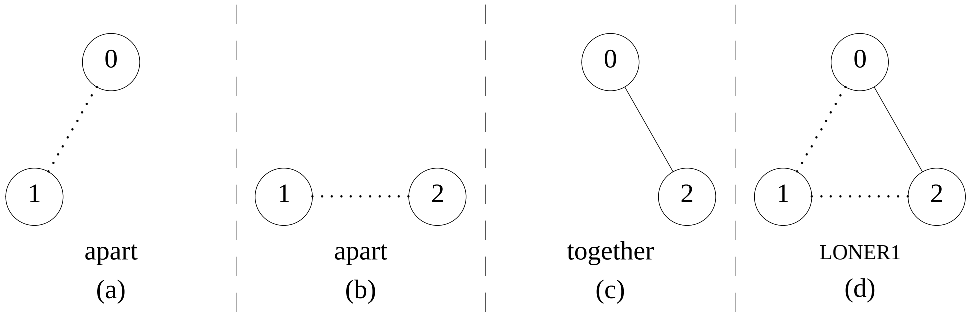

Figure 7-8:
Dyads Indicate Loner Placement:
(a)-(c) Dyads, (d) Conclusion

The combination of the three dyad results may not yield a three-process placement. Consider the dyad results shown in Figure 7-9.

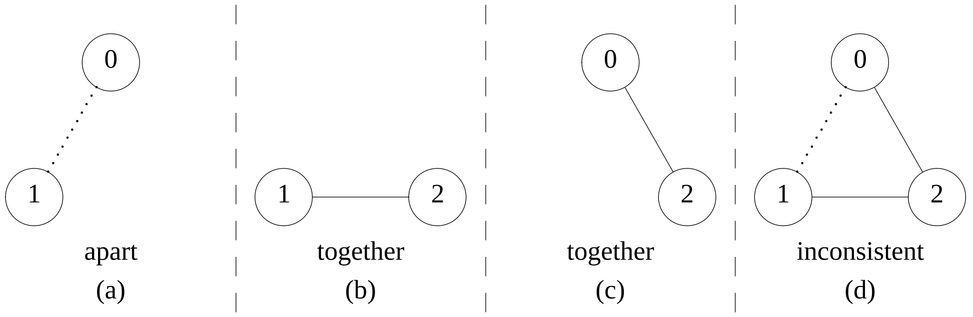

Figure 7-9:
Inconsistent Dyads:
(a)-(c) Dyads, (d) Inconsistent Result

There is no three-process configuration which keeps processes 0 and 1 apart, and 1 and 2 together, and 0 and 2 together. We call a combination of dyad results which does not yield a three-process configuration *inconsistent*.

Figure 7-10 lists all possible combinations of dyad results and the three-process configurations they indicate, or ‘‘inconsistent’’ if they do not yield a three-process configuration.

Dyad 0-1	Dyad 1-2	Dyad 2-0	3x3 Conclusion
apart	apart	apart	APART
apart	apart	together	LONER1
apart	together	apart	LONER0
apart	together	together	inconsistent
together	apart	apart	LONER2
together	apart	together	inconsistent
together	together	apart	inconsistent
together	together	together	TOGETHER

Figure 7-10:
All Possible Combinations of Dyad Results

There is exactly one combination of dyad results which yields each three-process configuration; there are three combinations which are inconsistent.

The type of three-process configuration indicated, APART, TOGETHER, or loner, seems to be determined by the number of apart and together dyad conclusions, as shown in Figure 7-11. It is only to determine which loner configuration is indicated that we need to see which specific dyads should be broken apart or kept together.

3 aparts	=	0 togethers:	APART
2 aparts	=	1 together:	a loner configuration
1 apart	=	2 togethers:	inconsistent
0 aparts	=	3 togethers:	TOGETHER

Figure 7-11:
Number of Apart and Together Dyad Results
Determines Type of Configuration

We call this approach *dyadic placement.* We have computed the dyad results and the three-process conclusion for each run in set U. There were no inconsistent results, and no loner results. For all runs in the first row of the grid (cells **ua** through **ue**), dyadic placement predicts that TOGETHER is the best configuration. For all other cases, dyadic placement predicts that APART is the best configuration. The predictions are divided by rows because each dyad result uses the PRED1 prediction based on *smextime*, and *smextime* is directly related to the number of execute actions, which varies by grid row. The actual simulation results are that in the **ua** runs, a loner is best; in all other runs, APART is best. Thus, dyadic placement is completely wrong in the first row, missing the loners in **ua** and missing the APARTs in **ub** through **ue**. Moreover, it is wrong in the worst possible way, choosing TOGETHER, which is the worst configuration. In the second row and below, dyadic placement is 100% correct, correctly predicting APART.

The first row of the grid is very unrealistic, so dyadic placement's failure in that row is not a serious failing. Dyadic placement is perfectly correct below the first row. However, since dyadic placement always chooses APART below the first row, it would be simpler to use the always-APART policy. Furthermore, the always-APART policy would be right for most of the first row, whereas dyadic placement is wrong for all of that row. Dyadic placement requires more data and more computation than the always-APART policy. In conclusion, dyadic placement is both less successful and more costly than always-APART.

7-2.2.8. Summary of Set U Results

In set U, as in 2x2 set N, APART is best in all cases except the single most high-communication cell. Even there, APART is only about 4% slower than the fastest configuration.

In almost every run, APART is fastest, and the three loner-type configurations are second-, third-, and fourth-fastest. In every run, TOGETHER is slowest. On average, APART is about 36% faster than the fastest loner; there is about 11% difference between the fastest and the slowest loner; and TOGETHER is about 51% slower than the slowest loner. Altogether, the slowest configuration is a mean of 98% slower than the fastest. We conclude that not only is there significant potential benefit in avoiding the worst configuration, but also in selecting the very best configuration. These benefits are in terms of percentages of response time, so the benefits will be more significant for large, long-running jobs (assuming that these results apply to much larger jobs).

If real processes collocated on a processing node are less similar to each other than our experimental processes, they may be better able to overlap use of the node's resources. If so, the differences in response times between configurations with larger and smaller numbers of collocated processes may be smaller than the differences we observed. We believe, however, that many or most distributed programs have child processes which are similar to each other.

The maximum number of processes collocated on any processor is a major influence on the response time of a configuration. Two 1-collocated configurations (2x2 APART and 3x3 APART) have similar response times and two groups of 2-collocated configurations (2x2 TOGETHER and the 3x3 loners) have similar response times. In both cases the three-process configurations have slightly larger response times, 5.38% and 1.72%, respectively; the increase is probably partly due to additional time spent waiting for messages to arrive, and partly due to the fact that the 2x2 percentages are computed relative to TOGETHER, usually the larger value, while the 3x3 percentages are computed relative to the fastest configuration, the smallest value. Based on the two-process and three-process experiments, the mean difference in response time between a 1-collocated and a 2-collocated configuration is about 30-50%, although individual differences can be from less than 1% to almost 100%. Based only on the three-process experiments, the mean difference between a 2-collocated and a 3-collocated configuration is about 51-62%, and the mean difference between a 1-collocated and a 3-collocated configuration is about 98%.

Always-APART is a very simple placement policy in which the APART configuration is always used; always-APART would be easy to implement and fast to execute. In set U, always-APART would pick the fastest configuration in every run except the unrealistic **ua** runs. Even in **ua**, the response time of APART is only about 4% slower than the fastest response time.

In dyadic placement, the PRED1 technique based on *smextime* is used on each pair of processes, or dyad. The dyad results are then combined to get a three-process placement (or an inconsistent result). In set U, dyadic placement choses TOGETHER for all runs in the top row of the grid; it choses APART in every other run; there are no loners or inconsistent results. These choices are completely wrong for the top row, and completely correct for the rest of the grid. Always-APART does better than dyadic placement, and it would run faster and require less data.

7-2.3. Experiment Set V as 3x3

7-2.3.1. Introduction

Experiment set V is based on the zoomed-in grid and has three processes per trace, as explained in Section 7-1.3.3, above. In each cell of set V there are four traces, or runs. In the following sections we discuss the results from set V. We begin by examining the outcomes of the simulations: we examine the winning configurations in Section 7-2.3.2, we use the winning configurations to determine the transition zone boundary in set V in Section 7-2.3.3, and we examine the response times in Section 7-2.3.4. We then examine the relative percentage differences between the response times in Section 7-2.3.5. In Sections 7-2.3.6 and 7-2.3.7 we consider the always-APART and dyadic placement placement procedures. In Section 7-2.3.8 we present our conclusions concerning set V.

7-2.3.2. Winning Configurations

The winning configurations in set V are shown in Figure 7-12.

	IO-COMM					
	IO COMM	IO COMM	IO COMM	IO COMM	IO COMM	IO COMM
CPU	0 100	1 99	2 98	3 97	4 96	5 95
0	**vaa** 2 2 1 0 0 1 0 1 1 0 2 2 a a a a t t t t	**vab** 0 2 0 2 2 1 2 1 1 0 1 0 a a a a t t t t	**vac** 2 2 0 1 1 0 2 2 a a 1 0 0 1 a a t t t t	**vad** 2 1 1 a a 0 2 2 1 a a 1 0 2 0 0 t t t t	**vae** a a a 0 2 1 0 1 1 0 2 a 0 2 1 2 t t t t	**vaf** a a 2 a 0 0 a 0 2 1 1 1 1 2 0 2 t t t t
1	**vag** 1 2 1 2 0 1 a 1 2 a 2 a a 0 0 0 t t t t	**vah** 1 0 a 2 0 1 1 a a a 0 0 2 2 2 1 t t t t	**vai** 2 1 2 a 1 a a 2 a 0 0 0 0 2 1 1 t t t t	**vaj** 1 a a 1 a 1 2 a 2 2 1 2 0 0 0 0 t t t t	**vak** a a a a 0 1 0 0 2 2 1 1 1 0 2 2 t t t t	**val** a a a a 1 1 2 0 2 0 0 2 0 2 1 1 t t t t
2	**vam** a a a 1 0 2 1 2 2 0 2 a 1 1 0 0 t t t t	**van** a 0 a a 2 a 2 0 0 1 0 2 1 2 1 1 t t t t	**vao** a 1 a a 2 a 0 1 1 0 2 0 0 2 1 2 t t t t	**vap** a a a a 0 1 1 0 1 0 2 1 2 2 0 2 t t t t	**vaq** a a a a 0 2 0 1 2 0 2 0 1 1 1 2 t t t t	**var** a a a a 1 0 0 2 2 2 2 1 0 1 1 0 t t t t
3	**vas** a a a a 2 1 1 0 0 2 2 1 1 0 0 2 t t t t	**vat** a a a a 1 0 1 0 2 1 2 1 0 2 0 2 t t t t	**vau** a a a a 0 1 0 0 2 2 1 1 1 0 2 2 t t t t	**vav** a a a a 2 1 2 1 1 2 0 2 0 0 1 0 t t t t	**vaw** a a a a 0 2 1 0 2 0 0 1 1 1 2 2 t t t t	**vax** a a a a 1 2 1 2 0 1 0 0 2 0 2 1 t t t t
4	**vay** a a a a 0 1 2 0 2 0 1 2 1 2 0 1 t t t t	**vaz** a a a a 2 1 2 2 1 2 0 1 0 0 1 0 t t t t	**vba** a a a a 1 0 2 1 0 2 0 2 2 1 1 0 t t t t	**vbb** a a a a 0 1 1 2 1 2 0 1 2 0 2 0 t t t t	**vbc** a a a a 2 0 0 1 1 1 1 0 0 2 2 2 t t t t	**vbd** a a a a 0 0 1 1 1 2 0 2 2 1 2 0 t t t t
5	**vbe** a a a a 0 0 2 1 1 2 1 2 2 1 0 0 t t t t	**vbf** a a a a 2 0 2 2 0 2 1 0 1 1 0 1 t t t t	**vbg** a a a a 0 2 1 1 2 0 0 2 1 1 2 0 t t t t	**vbh** a a a a 2 1 1 1 1 2 0 0 0 0 2 2 t t t t	**vbi** a a a a 0 2 1 1 2 1 2 2 1 0 0 0 t t t t	**vbj** a a a a 0 1 1 2 2 0 2 1 1 2 0 0 t t t t

Figure 7-12:
Winning Configurations in Set V

There are four runs per cell, one column per run. The fastest configuration is at the top of the column, the slowest at the bottom.
Key:

Symbol	Configuration
a	APART
0	LONER0
1	LONER1
2	LONER2
t	TOGETHER

In most runs, APART is the fastest configuration. In most cells, APART is fastest in every run, but, as in set R, there is a region in the upper left in which APART is not fastest in every run. In the extreme upper left, APART is one of the worst configurations. Further right and further down, APART gets better and better, and soon it is always best.

TOGETHER is worst in every run without exception, even in the extreme upper left; the same result was seen in set U. We believe that TOGETHER will continue to be the worst configuration in all parts of the grid with larger numbers of processors and processes.

The three loner configurations appear in various orders and various positions. As in set U, usually they are the second, third, and fourth best configurations.

7-2.3.3. Transition Zone in Set V

In 2x2 set R, APART is the fastest configuration in most of the grid, but there is an area in the upper left of the grid in which APART is not always faster; this region is called the transition zone (see Section 6-2.3.1). In set R, the transition zone extended down to most of the 3% CPU row, though it moved up to above **rap** and **rar** in those columns.

In set V, there is no area where TOGETHER is best; in fact, TOGETHER is always the worst configuration. There is a small area of four cells in which APART is never best (**vaa**, **vab**, **vac**, and **vag**), and a subarea of two cells in which APART is always second worst (**vaa** and **vab**).

In set V, the transition zone has moved upward and a bit left as compared to the set R transition zone. Instead of extending down to the 3% CPU row, it goes only to the 2% CPU row, so that the set V transition zone is six cells smaller than the set R transition zone. This implies that the always-APART policy will be even more successful in 3x3 than in 2x2. If the transition zone continues to shrink as the number of processes grows, there will be very few cases in which APART is not the best configuration, and the always-APART policy will be more and more successful.

7-2.3.4. Response Times

As examples of the response times in set V, Figure 7-13 shows the response times of all five configurations for one of the four runs in each of the corner cells of the grid. The response times for run number 2 in each of the four corner cells are shown.

	IO-COMM		
CPU	IO 0 COMM 100	...	IO 5 COMM 95
0	**vaa-2**		**vaf-2**
	1,392		2,374
	1,399		2,410
	1,424	...	2,471
	1,461		2,547
	1,503		2,780
...	...	...	...
5	**vbe-2**		**vbj-2**
	7,307		6,169
	7,796		6,362
	8,010	...	6,776
	8,295		6,826
	9,327		7,617

Figure 7-13:
Response Times (in ms)
for Run #2 of the Corner Cells of Set V
Fastest Time on Top to Slowest on Bottom

The mean response time over the 720 simulations in set V is 4,441 ms, with standard deviation 1,855 ms; the fastest response time is 1,280 ms (LONER0 in **vaa-4**), the slowest is 9,377 ms (TOGETHER in

vbh-4).

7-2.3.5. Relative Differences

7-2.3.5.1. Introduction

In this section we discuss the differences between the response times of the five configurations simulated for each run. In the following sections we consider step differences and best/order differences. The definitions of these differences are given in Section 7-2.2.4, above.

7-2.3.5.2. Step Differences

We have computed the step differences for each run in set V, as we previously did for set U in Section 7-2.2.4.2. The mean of all step differences in set V, including differences between best and second-best, second-best and third-best, and so on, is 203 ms, with standard deviation 243 ms; the minimum step difference is 0.02 ms, the maximum is 1,464 ms. The mean relative percentage step difference is 4.44%, with standard deviation 4.37%; the minimum is 0.001% and the maximum is 21.06%.

Figure 7-14 shows the mean step differences for each step separately.

	Absolute Difference (ms)			
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	151	151	0.02	686
2nd-Best – 3rd-Best	86	100	0.02	431
3rd-Best – 4th-Best	113	126	0.46	621
4th-Best – Worst	463	310	41.67	1,464

	Relative Percentage Diff.			
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	3.00	2.31	0.001	11.33
2nd-Best – 3rd-Best	1.85	1.76	0.001	7.04
3rd-Best – 4th-Best	2.54	2.44	0.03	10.81
4th-Best – Worst	10.36	3.84	2.99	21.06

Figure 7-14:
Mean Step Differences for each Step

All of these figures are much smaller than the corresponding figures for set U. In the zoomed-in grid, there is less difference between the response times of different configurations, and it is less important to choose a configuration carefully. The mean step difference of 4.44% is quite small.

The clustering seen in set U is not so clearly seen in set V. There is a relatively large difference between the fourth-best response time and the worst, usually the difference between the slowest loner and TOGETHER. The difference between the best and second best, usually the difference between APART and the fastest loner, is still larger than the differences between the loners, but not that much larger.

7-2.3.5.3. Best/Order Differences

Figure 7-15 shows the mean best/order differences, over all 144 runs in set V.

	Absolute Difference (ms)			
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	151.31	150.50	0.02	686.02
Best – 3rd-Best	236.82	205.78	5.53	900.61
Best – 4th-Best	349.86	267.86	20.28	1,055.28
Best – Worst	813.03	558.11	110.19	2,426.90

	Relative Percentage Diff.			
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	3.00	2.31	0.001	11.33
Best – 3rd-Best	4.85	3.04	0.38	13.82
Best – 4th-Best	7.40	4.02	0.93	20.42
Best – Worst	17.76	7.12	5.62	40.69

Figure 7-15:
Mean Best/Order Differences in Set V

The differences are much smaller in set V than in set U. Except for the best/worst difference, the differences are fairly small, so it may not be worthwhile selecting the best configuration, depending on the time cost of making the selection. On the other hand, individual differences range up to 11% even for the best/second-best difference, so perhaps it would be worthwhile, again depending on the time cost. Furthermore, it is probably worth avoiding the worst configuration; given that some selection is to be made, the additional cost of identifying the best configuration may be small.

7-2.3.6. The Always-APART Policy in Set V

In this section we examine the success of the always-APART policy in set V. The always-APART policy is described in Sections 6-6 and 7-2.2.6.

In set V, there are a total of 144 runs; in 112 of those runs, or 77.78%, APART is the fastest configuration; in 32 runs, or 22.22%, APART is not fastest, so the always-APART policy picks the wrong configuration. The difference between the response time of the best configuration and the response time of APART is the *APART penalty*. This penalty is often zero since APART is often the fastest configuration. The mean APART penalty over all runs in set V is 7.56 ms, or 0.42% of the fastest response time. The largest penalty is 84 ms, or 6.22%.

These figures over set V are not very meaningful, though, because the right and bottom boundaries of the zoomed-in grid are arbitrary. The transition zone provides a naturally defined region within set V, so we consider the success of the always-APART policy inside and outside the transition zone.

Outside the transition zone, always-APART performs perfectly. The transition zone is defined as the area in which APART is not always best, so, by definition, outside the transition zone always-APART is right 100% of the time and the APART penalty is always zero. In set V, the transition zone consists of cases **vaa-vaj** and **vam-vao**, which is 13 cases and 52 runs. In 20 of the 52 runs, or 38.46%, APART is the winner. The mean APART penalty in the transition zone is 21 ms, or 1.16% of the fastest response time; this is a very small difference. Penalties are largest in the upper left-hand corner and grow generally smaller further to the right and further down. The largest penalty is 84 ms, or 6.22% (the same as for all of set V, of course).

We think it very unlikely that any real, useful program would fall in the transition zone. Outside the transition zone, APART is always best by the definition of the zone. Even in the transition zone, always using APART will be right a significant portion of the time. The average penalty for using APART instead of the best possible configuration is only about 1%, so even when APART is not best, it is only slightly worse than the best; the worst penalty for using APART instead of the best configuration is only 6.22%. These penalty areas and penalty sizes are quite small. Simply always using the APART configuration is even more successful for 3x3 than for 2x2.

7-2.3.7. Dyadic Placement in Set V

We introduced dyadic placement in Section 7-2.2.7. In set U, dyadic placement chooses TOGETHER for all runs in the top row of the full grid and APART for all other runs. There are no inconsistent outcomes.

In set V, dyadic placement again choses TOGETHER for every run in the zero percent CPU row, cells **vaa-vaf** (see Figure 7-12). In cells **vag-vap** dyadic placement choses a mixture of APART, TOGETHER, and loners, and there are a number of inconsistent outcomes. In **vaq-vbj**, dyadic placement always choses APART.

Over all of set V, dyadic placement correctly chooses the best configuration 98 times out of 144 runs, or in 68.06% of the runs. The 98 hits include 95 which are APART and 3 which are LONER1, thus most of the successes are cases in which dyadic placement correctly chooses APART.

We again use the transition zone as a naturally defined region. The transition zone is the region in which it is most difficult to make good placement decisions, so the success rate of dyadic placement in the transition zone will be a lower bound on the performance of dyadic placement. For purposes of this investigation, we define the transition zone as extending from **vaa** through **vap**; cells **vak**, **val**, and **vap** are included because dyadic placement chooses configurations other than APART in those cells. This transition zone includes 16 cases and 64 runs.

In this transition zone, dyadic placement successfully predicts the best configuration in 18 runs, or 28.13% of runs. In 15 of those runs the best configuration is APART, in three runs it is LONER1.

Both over set V as a whole and inside the transition zone, the always-APART policy has a higher success rate than dyadic placement, about ten percentage points higher. In addition, always-APART would be easier to implement and faster to run.

7-2.3.8. Summary of Results from Set V

In most runs in set V, APART is the fastest configuration; TOGETHER is slowest in all runs. There is a transition zone in the top and upper left of the grid in which APART is not always the best configuration; below and right of the transition zone APART is always the fastest configuration. The transition zone in set V is smaller by six cells than the transition zone in 2x2 set R. The differences between the response times of the configurations are much smaller in set V than in set U. The mean step difference is only 4.44%, but some step differences are as large as 21%. The clustering observed in set U is present, but it is much less pronounced. The mean best/worst difference is about 18%; the largest best/worst difference is about 41%.

The always-APART policy is correct in selecting APART in about 79% of the runs in set V; the mean APART penalty is 0.42% of response time, the largest penalty is about 6%. Outside the transition zone, always-APART is, by definition of the zone, perfect. Inside the transition zone, always-APART selects the fastest configuration in only about 38% of the runs, but the mean APART penalty is still only 1.16%.

Over all of set V, dyadic placement selects the fastest configuration in about 68% of the runs; almost all of these are APART. In the transition zone, dyadic placement selects the fastest configuration in only about 28% of runs. Always-APART has a higher success rate, and is easier to implement and faster to run.

7-2.4. Summary and Conclusions Concerning 3x3

We have already summarized the results of the set U and set V analyses. In this section we very briefly summarize the overall results of our analyses of the 3x3 situation.

The APART configuration is the fastest 3x3 configuration; on average over the full grid it is about 36% faster than the next fastest configuration. Only in runs with 3% or fewer CPU execute actions is APART not always the fastest configuration. Even in such extreme cases, APART is only a few percent slower than the fastest configurations (about 4% for the **ua** runs, an average of 1.16% over the set V transition zone); APART is never more than about 6% slower than the fastest configuration.

The response times of the configurations cluster by type. The difference between the response time of the fastest loner-type configuration and the slowest is only about 11% on average over the full grid. This is less than the average 36% difference between APART and the fastest loner, and much less than the average 51% difference between the slowest loner and TOGETHER. TOGETHER is always the slowest configuration, even in the upper left corner of the zoomed-in grid; it is, on average, 98% slower than the fastest configuration.

The always-APART policy works very well in 3x3 situations. It is nearly perfect in set U, making correct selections in every case but the extreme upper left cell, **ua**. In set V, it is perfect outside the transition zone; inside the transition zone it selects the correct configuration in only about 38% of the runs, but the configuration it picks is, on average, only 1.16% slower than the fastest configuration. The largest APART penalty is about 6%.

Dyadic placement is a placement policy in which the PRED1 technique based on *smextime* is used on each pair of processes, or dyad. The dyad results are then combined to get a three-process placement. Dyadic placement performs moderately well, but not as well as always-APART. In addition, always-APART is much simpler, meaning that it would be easier to implement, would not involve collecting and storing information about programs or processes, and would execute faster at placement time.

7-2.5. Summary and Conclusions Concerning 2x2 and 3x3

In both 2x2 and 3x3 situations, APART is overwhelmingly the fastest configuration. Only when there are less than 4% CPU execute actions is APART not fastest, and even then APART is only about 4% slower than the fastest configuration. In both situations, moderately successful predictions of the fastest configuration can be made based on CPU service demand, but in both it is faster and more effective to always use APART.

The transition zone (the region in which APART is not always the best configuration) is slightly smaller in 3x3 set V than in 2x2 set R. If it continues to shrink as the number of processes grows, the always-APART policy make be even better for larger numbers of processes.

Generalizing, it appears that an APART configuration will be fastest in nearly any NxN situation, the exceptions being unrealistic cases in which the amount of communication is huge and the amount of computation small.

APART configurations use all available processing nodes, and spread the processes out as much as possible. Generalizing from this, we suppose that in NxM situations the fastest configurations will be those which make use of all M processing nodes, thus spreading out the processes as much as possible. For example, arrangements of the form 1:1:...:1:1 should be fastest when there are N processing nodes, and arrangements of the form 2:1:1:...:1:0 should be fastest when there are $N - 1$ processing nodes.

On the other hand, the maximum number of processes collocated on any processing node seems to determine the response time of a configuration. The two APART configurations, each with one process per processor (1-collocated) have similar response times; and the two groups of configurations with no more than two processes per processor (2-collocated), 2x2 TOGETHER and the 3x3 loners, also have similar response times. This result also indicates that an APART configuration should be fastest in any NxN situation, but it does not indicate whether an arrangement of the form 2:1:1:...:1:0 or one the form 2:...:2:0:...:0 would be fastest when there are $N-1$ processing nodes. Indurkhya, Stone, and Xi-Cheng's [ISX86] and Nicol's [Nic89] results, described in Section 2-3.2, indicate that the 2:...:2:0:...:0 arrangement is likely to be faster. This issue will be discussed further in Chapter 8.

Although the 1-collocated and 2-collocated configurations have similar response times, in both cases the three-process configurations have slightly larger response times, 5.38% and 1.72% larger, respectively; the increase is probably partly due to additional time spent waiting for messages to arrive, and partly due to the fact that the 2x2 percentages are computed relative to TOGETHER, usually the larger value, while the 3x3 percentages are computed relative to the fastest configuration, the smallest value. Based on the two-process and three-process experiments, the mean difference in response time between a 1-collocated and a 2-collocated configuration is about 30-50%, although individual differences can be from less than 1% to almost 100%. The mean difference is much less than double, indicating that there is considerable overlap in processes' use of the resources of a processing node. Based only on the three-process experiments, the mean difference between a 2-collocated and a 3-collocated configuration is about 51-62%, and the mean difference between a 1-collocated and a 3-collocated configuration is about 98%. Overall, and not terribly precisely, it appears that adding a processing node improves response time by very approximately 50%.

7-3. Three Processes with Two Processors (3x2)

7-3.1. Introduction

When there are three processes to execute but only two processing nodes available, a 3x2 situation, where should the processes be placed? Which factors affect the response times of the configurations? How large are the differences between response times? Can the best configuration be identified? How should placement decisions be made? These questions pertaining to the the 3x2 situation are the subject of this section.

This is the first time we encounter a situation where there are fewer processors than processes yet there is still a choice of configurations. With only two processors available, the 3x3 APART configuration is not possible. TOGETHER is the worst configuration in every run in sets U and V, so we can eliminate it from consideration. This leaves the three loner-type configurations, LONER0, LONER1, and LONER2, as candidates for the fastest 3x2 configuration. The question in a 3x2 situation, then, is which loner configuration will be fastest, in other words, which process should be the loner process.

We have seen in the 2x2 and 3x3 situations that, if N processors are available, the best response time is obtained by using all of them in an APART configuration, and we conjectured that if N-1 processors are available, the best response time is obtained by using a configuration with no more than two processes on any processing node and by spreading out the processes as much as possible. For the 3x2 situation, both conjectures indicate the loner configurations, which are in the 2:1:0 arrangement.

If the differences between the response times of the loner configurations are small, it does not matter which one is used. We could just pick one randomly, or according to convenience, rather than spending time at placement time trying to pick the very fastest one. Our first step, then, will be to examine the differences in the response times of the loner configurations; we do this in Section 7-3.2 for sets U and V. After examining the differences in response times, we will investigate two approaches to selecting a loner process: *matching* and *attraction/repulsion.* Although we usually present the full grid first, followed by the zoomed-in grid, in this section we will use a somewhat different order. We present the two placement procedures for zoomed-in set V first, in Sections 7-3.3 and 7-3.4; it is more difficult to make accurate predictions in the zoomed-in grid, so the set V analyses should be a more rigorous test of the two techniques. We summarize our conclusions concerning the two techniques and finding the best loner in set V in Section 7-3.5. We then briefly examine the performance of the two techniques in full grid set U in Section 7-3.6. Section 7-3.7 is a summary of our conclusions concerning selecting the best 3x2 configuration.

Recall that the analyses of the 3x2 situation use the same traces and simulation results as the 3x3 analyses, except that the APART simulation results are not used.

7-3.2. Response Time Differences Between Loners

7-3.2.1. Loner Response Time Differences in Set U

It so happens that in set U the three loner configurations in each run always occur in sequence in the order of configuration response times (they are second-, third-, and fourth-best in all runs except **ua-1** and **ua-2**, where they are best, second-best, and third-best), so the differences between loners are the same as two of the step differences (see Section 7-2.2.4.2). Figure 7-16 shows the summary statistics for the relative percentage differences between the response times of (1) the best loner and the second-best loner; (2) the second-best loner and the worst loner; and (3) all loner configurations, the union of (1) and (2).

	N	Mean	Std. Dev.	Smallest Diff.	Largest Diff.
Best - 2nd-Best Diff.	52	5.07%	4.27%	0.02% (**um-2**)	19.82% (**ux-1**)
2nd-Best - Worst Diff.	52	6.00%	5.57%	0.42% (**ua-1**)	27.22% (**uw-2**)
Diffs. Between All Loners	104	5.54%	4.98%	0.02% (**um-2**)	27.22% (**uw-2**)

Figure 7-16:
Summary Statistics for Relative Percentage Differences
Between Loner Configuration Response Times in Set U

The mean difference between loner configurations is 5-6%, a small but significant fraction of the response time, and the differences get as large as 27%. The distribution of relative percentage differences is shown in Figure 7-17.

Range of Relative Differences	Number of Runs with Relative Difference in Range
0.00 - 1.99	29
2.00 - 3.99	21
4.00 - 5.99	19
6.00 - 7.99	10
8.00 - 9.99	5
10.00 - 11.99	8
12.00 - 13.99	6
14.00 - 15.99	2
16.00 - 17.99	1
18.00 - 19.99	1
20.00 - 21.99	0
22.00 - 23.99	1
24.00 - 25.99	0
26.00 - 27.99	1

Figure 7-17:
Distribution of Loner Relative Percentage Differences
in set U

Most of the differences are small, but there are a few large ones.

It is worth a certain amount of effort and placement decision-making delay to try to obtain a mean 5-6% improvement in response time. This is especially true if the programs are long-running; a 5% speedup in a program which runs for 20 hours is an hour faster response time.

7-3.2.2. Loner Response Time Differences in Set V

In set V, the loners are not all consecutive in the order of best-to-worst response times; we have computed the loner response time differences. Figure 7-18 shows the summary statistics for the relative percentage differences in a manner similar to that of Figure 7-16.

	N	Mean	Std. Dev.	Smallest Diff.	Largest Diff.
Best - 2nd-Best Diff.	144	1.86%	1.74%	0.001% (**vac-2**)	7.04% (**vas-4**)
2nd-Best - Worst Diff.	144	2.54%	2.44%	0.003% (**vag-1**)	10.81% (**vbc-1**)
Diffs. Between All Loners	288	2.20%	2.15%	0.001% (**vac-2**)	10.81% (**vbc-1**)

Figure 7-18:
Summary Statistics for Relative Percentage Differences
Between Loner Configuration Response Times in Set V

The mean differences between loner configurations in set V are only about 2%, although the largest differences are almost 11%.

7-3.3. Matching

7-3.3.1. Introduction, Description, and Procedure

In this section, we introduce the *matching technique* for identifying the best loner process, and therefore the fastest loner configuration, which is also the fastest 3x2 configuration. We then apply this technique in set V and analyze the results.

Matching is based on the hypothesis that the best loner process will be the one with the largest or the smallest amount of some measurable quantity. Specifically, our hypothesis is that the best loner process is the one with the largest elapsed CPU time, or possibly the one with the smallest communication waiting time. We sort the processes of each run into order according to a number of these quantities, and compare the sorted orders to the order of best processes according to response time.

In the remainder of this section we describe the matching technique in more detail, including two variations: *eql* and *noeql.* In Section 7-3.3.2 we analyze the results of using the matching technique in set V. We search for trends in the way the matching results change by comparing the results inside and outside the transition zone in Section 7-3.3.3, and by examining the results in various *slices* of the grid in Section 7-3.3.4. In Section 7-3.3.5 we check how often matching selects the worst loner configuration, and in Section 7-3.3.6 we experiment with using two variables to do matching. The final section on matching, Section 7-3.3.7, summarizes the results and presents our conclusions.

7-3.3.1.1. Loner Order

The processes in a distributed program can be ordered according to the response times of the loner configurations. If LONER2 is the best (fastest) loner configuration, LONER1 the next best, and LONER0 the worst of the loner configurations, then the *loner order* is 2-1-0.

7-3.3.1.2. Variable Order

One way to try to predict the best loner process is to pick the process that has the least, or the most, of some measurable quantity (the *variables* of Chapter 6). For example, the process with the least communication activity might be the best loner, or the process with the most CPU activity might be the best loner.

The processes can be ordered by increasing or decreasing values of some variable. For example, the process order according to increasing bytes of data sent in messages might be 2-0-1, meaning that process 2 sent the fewest bytes of data, process 0 the next largest number of bytes, and process 1 the most.

7-3.3.1.3. Comparing the Loner and Variable Orders

Comparing the loner order to an increasing variable order has three possible outcomes of interest: (a) the orders match completely, or *all-match,* that is, the process with the least of the quantity is the best loner process, the process with the middle amount is the second-best loner, and the process with the most of the quantity is the worst loner; (b) only the best matches, or *only-best-match,* that is, the process with the least of the variable is the best loner process, but the process with the middle amount is not the second-best loner, and so on; or (c) the orders do not match at all. In our example, the loner order is 2-1-0 and the increasing quantity order is 2-0-1, so only the best loner matches. Comparing the increasing quantity order to the loner order is called *least-is-best matching.*

Similarly, the decreasing quantity order can be compared to the loner order; this is called *most-is-best matching,* and the same three outcomes are possible. In our example, the decreasing quantity order would be 1-0-2. Comparing to the loner order, 2-1-0, the result of most-is-best matching is no match. For each variable we try both *directions:* least-is-best matching and most-is-best matching.

To summarize the results over the whole experiment set, we count the number of times the best loner is successfully matched, that is, the sum of all-matches and best-only-matches, for each variable, for each direction of ordering. From this *hit count,* we compute the percentage of runs in which each variable/direction hits.

7-3.3.1.4. Variables Used in Matching

There are a number of variables which might produce good matching results. Variables drawn from the process trace summaries (see Section 5-5) pertain to the processes themselves, and are independent of configuration, processor speed, and other simulation parameters. The process summary variables used in matching are shown in the upper part of Figure 7-19. Variables drawn from the simulation outputs (see Section 5-2.5) do depend on the configuration, the processor speeds, the amounts of CPU time required for sends and receives, and other system parameters. The simulation output variables used are shown in the lower part of Figure 7-19. Note that these variables are *elapsed* times, not service times.

Process Summary Variables	
naction	number of actions in the process trace
nexec	number of execute actions in the process trace
sumexec	sum of CPU service times specified in execute actions
nio	number of I/O actions
sumio	sum of bytes read or written in I/O actions
nsend	number of send actions
sumsend	sum of bytes of data sent in messages
nrcv	number of receive actions

Simulation Output Variables	
cputime	actual elapsed CPU time (elapsed real time spent getting CPU service)
iotime	actual elapsed I/O time (elapsed real time spent doing I/O)
commtime	actual communication waiting time (elapsed time waiting for messages to arrive)

Figure 7-19:
Variables Used in Matching

The basic matching procedure is performed in both directions (most-is-best and least-is-best) for each of these variables in every run in set V.

7-3.3.1.5. Configurations Used

The values of the simulation output variables depend on the configuration in which the program is executed, so the matching success rate might be different depending on which configuration's simulation outputs are used. For each run, we ran the full matching procedure twice, using the simulation outputs from the APART and TOGETHER configurations. TOGETHER and APART are the extreme configurations, and the results from the loner configurations should fall somewhere between them. There is no reason to think that using any of the loner configurations would be more successful than using one of APART or TOGETHER. To use matching in a placement procedure might require running (or simulating) the program. It would be easiest to run the program TOGETHER, since only one processing node would be required, so we hope that the success rates using TOGETHER are at least as good as the APART success rates.

7-3.3.1.6. Process Order when Variable Values are Equal: Eql and Noeql

When two or more processes have the same value for some variable, it is not clear what the variable order should be, and the variable order will have a direct effect on the matching hit rate. We use two different approaches to determine the variable order in such cases: *eql* and *noeql.* One approach overstates the success of matching, the other understates it, so the actual success rate should be between the eql and noeql hit rates. In the following sections we describe the eql and noeql approaches.

7-3.3.1.6.1. Noeql

The simpler approach is to disallow any matches when two or more processes have the same value for some variable. In such a case, no variable order is generated, so no match with the loner order is possible and no matching success will be counted. While this is simple, it is too simple; it might still be possible to check for a best-only match. For example, if one process has three I/O operations and the other two both have seven, then for least-is-best it is still possible to check whether the process with three operations is the best loner. Even if it is not possible to tell which process has the least (or most) of a variable, it might be possible to eliminate the worst process, leaving a choice between two equal-valued processes. Only when all three processes have the same value is it really not possible to tell anything.

This approach therefore understates the success of matching. We call this approach *noeql,* for ‘‘no matches when there are equal values.’’

7-3.3.1.6.2. Eql

The second approach to handling equal values is to generate every process order allowed by the values and check them all for matches. For example, if process 0 has 395 actions and processes 1 and 2 each have 402, the orderings allowed are 0-1-2 and 0-2-1 for least-is-best, or 2-1-0 and 1-2-0 for most-is-best. If the loner order is 0-1-2, then we would count an all-match and a best-only match for least-is-best, and no matches for most-is-best. When two processes have equal values of a variable, two orderings are possible. When all three processes have equal values, six orderings are possible, which is all possible orderings of three processes; one of the six will be an all-match, and another will be a best-only match, for each of least-is-best and most-is-best. We call this approach of generating all possible orderings when there are equalities the *eql* approach.

There are a number of cases and a number of variables for which the variable must have value zero for all three processes. All cases in the top row of the grid (**vaa**-**vaf**) have zero percent execute actions, so in that row the variables *nexec* and *sumexec* are always zero in all processes (the variable *cputime* is not always zero, because I/O’s, sends, and receives require some CPU service). Similarly, the cases in the left-hand column of the grid (**vaa**, **vag**, **vam**, **vas**, **vay**, and **vbe**) all have zero percent I/O actions, so in that column the variables *nio*, *sumio*, and *iotime* are always zero in all processes. With all three variables equal, all six orderings would be generated, and an all-match and a best-only match for each of most-is-best and least-is-best would be guaranteed, but these matches would not represent any real success of the matching approach. To eliminate these false successes, we do not count any matches when the value of a variable is zero in all three processes. The eql approach still overstates the success of matching, but, with this adjustment, the extent of the overstatement is reduced.

After elimination of the all-zero cases, there are only a few runs in which a variable has the same value in all three processes; these are listed in Figure 7-20.

Run	Variable	Configuration(s)
vag-2	*naction*	APART and TOGETHER
vak-1	*nexec*	APART and TOGETHER
vbc-3	*iotime*	APART
vbf-1	*nio*	APART and TOGETHER

Figure 7-20:
Runs, Variables, and Configurations
in which All Three Processes have the Same Value

When the values are drawn from the APART configuration simulation outputs, there are 108 instances in which two processes have the same variable value. Using the TOGETHER configurations, there are 82 instances. These instances occur in every case except **vaa**.

7-3.3.1.6.3. Comparing Eql and Noeql

The noeql approach understates the success of matching, while the eql approach overstates it. We have computed the results for both approaches, and presumably the true results are somewhere between. We believe that noeql understates quite a bit, since there is no match whenever two or three values are equal. Eql overstates less, especially with the all-three-zero elimination. When two values are equal, an all-match or a best-only-match may or may not occur. Only when the variable values for all three processes are equal is a match guaranteed, and that happens only three or four times.

An alternative to eql and noeql in dealing with equal values would be to count only half a match for a match when two processes have equal values, and only one-sixth of a match for a match when three processes have equal values. We have not done this.

7-3.3.1.7. Performing Matching

The basic matching routine is performed four times for each run in set V, for each combination of the APART and TOGETHER simulation results and the eql and noeql approaches. The basic matching routine does both most-is-best and least-is-best matching. Each execution of the matching routine produces, for each variable and each direction, an indication of whether that variable and direction produced no match, a best-only-match, or an all-match.

The per-run results are summarized over the whole experiment set. There are separate summaries for each of the combinations of APART and TOGETHER, eql and noeql. Each summary includes six tables, showing the success rates for each variable and direction, for each of all-match, best-only-match, and best-match (best-match being the sum of all-match and best-only-match). For example, Figure 7-21 shows the most-is-best all-match results from the eql APART summary.

Set V Eql APART
Most-is-Best All-Match

Variable	Count	% of 144
nexec	57	40%
nrcv	25	17%
sumsend	18	13%
sumexec	45	31%
cputime	53	37%
nio	30	21%
sumio	24	17%
iotime	36	25%
naction	30	21%
nsend	22	15%
commtime	10	7%

Figure 7-21:
Matching Results:
Set V Eql APART Most-is-Best All-Match

Having described how matching is done, and the type of results produced, we now turn to the results of matching in set V.

7-3.3.2. Matching Results in Set V

In this section, we discuss the results of matching in set V. The all-match results from all four variations are presented first, then the best-match results from all four variations. (The best-only results are not presented; they are used only to compute the best-match results.) Finally, we present some conclusions drawn from the matching results.

Our objective is to identify the variables and directions which are most successful at selecting the best loner process, which in turn reveals the best 3x2 configuration. All percentages shown are percentages out of the 144 runs in set V.

7-3.3.2.1. All-Match Results

The success rates of the variable/directions for eql APART range from 6.25% to 39.58% (Figure 7-21, above, shows the eql APART most-is-best all-match results). Only a few of the variables have success rates greater than 30%. We are interested in the most successful variables, so let us make 30% our cut-off threshold; any variables with success rates of 30% or less will be ignored. For eql APART, four variable/directions make the cut-off; their success rates are shown in Figure 7-22, below.

The hit rates for the simulation output variables are higher in eql TOGETHER matching than in eql APART matching; the differences range from zero to almost ten percentage points higher. The variables from the process summaries have the same hit rates, of course. The same four variable/directions have success rates above the 30% cut-off in eql TOGETHER and eql APART; see Figure 7-22.

The *inverses* of some of the successful variable/directions (same variable, opposite direction) are themselves somewhat successful. For example, least-commtime successfully predicts the loner order in about 48% of the runs, but most-commtime succeeds in about 6% of the runs. Similarly, most-cputime succeeds in 43% of runs and least-cputime in 8%. This is troubling, as it might indicate that the variable/directions select the worst loner more often than we would like; we will look into this in Section 7-3.3.5, below.

In noeql matching, as in eql, matching using data from the TOGETHER simulations is more successful than that using APART. The same four variable/directions have the best success rates in noeql as in eql. Figure 7-22 shows the success rates of the four most successful variable/directions for all four matching variations.

Variable and Direction	eql		noeql	
	TOGETHER	APART	TOGETHER	APART
least-commtime	48%	39%	48%	39%
most-cputime	43%	37%	43%	37%
most-nexec	40%	40%	26%	26%
most-sumexec	31%	31%	31%	31%

Figure 7-22:
All-match in set V,
the Most Successful Variable/Directions

For both eql and noeql, the highest success rates are for least-commtime and most-cputime using the TOGETHER configuration.

To summarize, for all-matches in all of V, the two most successful variables are (1) least communication waiting time in a test run executed in TOGETHER configuration, which succeeds in predicting the complete loner order about 48% of the time, and (2) most elapsed CPU time in a test run executed in TOGETHER configuration, which succeeds in predicting the complete loner order about 43% of the time. To be able to predict the full loner order in close to 50% of the runs is quite good, but 50% is much too low for a practically useful placement procedure. Fortunately, a placement procedure only has to select the best loner.

7-3.3.2.2. Best-Match Results

A best-match occurs when the best loner predicted by a high or low value of a variable matches the actual best loner, regardless of whether or not the rest of the order is matched. The number of best-matches is the sum of all-matches and best-only matches, so the success rates for best-match are higher than those for all-match. We raise the cut-off point to 50%; variable/directions with success rates of 50% or less will not be discussed.

The same four variable/directions that were most successful in all-match are again most successful in best-match. Only three of them, however, have success rates greater than the cut-off; these are shown in Figure 7-23.

Variable and Direction	eql		noeql	
	TOGETHER	APART	TOGETHER	APART
least-commtime	64%	59%	64%	59%
most-nexec	63%	63%	37%	37%
most-cputime	59%	53%	59%	53%

Figure 7-23:
Best-match in set V,
the Most Successful Variable/Directions

In best-match, as in all-match, matching using TOGETHER is more successful than matching using APART. In best-match, however, the sizes of the differences are smaller.

Again there is some success in using the inverse variable/directions: least-commtime 64%, most-commtime 12.5%; most-cputime 59%, least-cputime 15%.

To summarize, for best-matches in all of V, the two most successful variables are the same as for all-match. They are: (1) least communication waiting time in a test run executed in TOGETHER configuration, which succeeds in predicting the best loner about 64% of the time, and the complete loner order 48% of the time, and (2) most elapsed CPU time in a test run executed in TOGETHER configuration, which succeeds in predicting the best loner about 59% of the time, and the complete loner order about 43% of the time.

7-3.3.2.3. Summary

The matching success rate of about 64% is rather lower than the 80-82% success rate achieved in 2x2 by the PRED1 predictions using *smextime* (see Section 6-8.3).

Our model of distributed programs involves processes that are quite similar. We believe that in scientific computing, the processes in programs will be quite similar, and we believe that most real distributed programs will be scientific programs. Matching achieved 64% success in picking the best loner out of three processes that are intentionally similar; in circumstances where the processes are less similar, the matching success rates should be higher. On the other hand, we believe that most scientific programs will fall into the lower left part of the full grid, while set V, investigated here, falls into the upper left part.

Matching using data from TOGETHER is more successful than matching using data from APART, both in terms of better success rates and lower success rates of inverse variable/directions.

The success of the inverse variable/directions might indicate a problem; we investigate this in Section 7-3.3.5.

The success rates of matching in set V are only moderately high. In an effort to better understand the success rates of matching, we divided set V into subsets in a number of different ways, and analyzed and compared the success rates in each subset. These divisions into subsets and the results of the subset analyses are the subjects of the next few sections.

7-3.3.3. Matching Based on the Transition Zone

7-3.3.3.1. Introduction

The modest success rates for matching in set V may be the result of low success rates in the transition zone combined with high success rates outside the transition zone. In this section, we check this hypothesis by separately analyzing the success of matching inside and outside the transition zone.

The transition zone is the set of cases in which APART is not always the best configuration. For this analysis we define the transition zone to be cases **vaa-vao**; these 15 cases include 60 runs. There are 21 cases, **vap-vbj**, including 84 runs, outside the transition zone. (Strictly speaking, cases **vak** and **val** should not be considered part of the transition zone, but for simplicity of computation, we have included them; see Section 7-2.3.3 and Figure 7-12.)

Matching has to do with which loner configuration is best, which is second-best, and so on, whereas the transition zone is defined in terms of the success of the APART configuration. There is no *a priori* reason to think that the two have anything to do with each other. We expected that the success rates in the transition and non-transition zones would be similar to each other, and similar to the success rates for set V as a whole.

7-3.3.3.2. Results

The highest all-match and best-match success rates (using TOGETHER data and the eql approach) inside and outside the transition zone of set V, and in set V as a whole, are shown in Figure 7-24.

All-Matches			
Variable/Direction	Inside Transition Zone	Set V as a Whole	Outside Transition Zone
least-commtime	47%	48%	49%
most-cputime	32%	43%	51%
most-nexec	28%	40%	48%
most-sumexec	25%	31%	36%
most-iotime	33%	22%	14%
most-nio	28%	21%	16%

Best-Matches			
Variable/Direction	Inside Transition Zone	Set V as a Whole	Outside Transition Zone
least-commtime	60%	64%	67%
most-cputime	48%	59%	67%
most-nexec	55%	63%	69%
most-sumexec	37%	47%	55%
most-iotime	45%	37%	31%
most-nio	43%	37%	32%

Figure 7-24:
Highest all-match and best-match success rates
using TOGETHER and eql
inside and outside the transition zone of set V
and in set V as a whole

Compared to the results for set V as a whole, the results for the transition zone range from 11 percentage points better to 28 percentage points worse; the results for outside the transition zone range from eight percentage points better to eight percentage points worse. Generally, the success rates are lower in the transition zone and higher outside the transition zone; only for the two I/O-based variable/directions, most-iotime and most-nio, is this reversed. In all cases, the results for set V as a whole are between the inside and outside results, as they ought to be.

Least-commtime is the most successful variable/direction in the transition zone, for both all-match and best-match. Outside the transition zone, three variable/directions have similar success rates: least-commtime, most-cputime, and most-nexec.

Outside the transition zone, the differences between the TOGETHER and APART success rates are large, up to 12 percentage points for all-match using most-cputime. Also, outside the transition zone there is a big difference between the eql and noeql results for some of the variable/directions; for most-nexec best-match the difference is 20 percentage points.

7-3.3.3.3. Conclusions and Discussion

It is not clear why the transition zone should generally be harder for matching to deal with than the area outside the transition zone, or why the reverse should be true for the I/O-based variable/directions. It is possible that success rates are lower in the upper left of the grid, and higher toward the lower right. In that case, any upper left region would have a lower success rate than a region lower down and further to the right; the transition zone might have nothing to do with it. If that were true, would the success rates continue to rise further down and right, even beyond the edges of the zoomed-in grid? These questions are addressed in the following section on slices (Section 7-3.3.4).

Program characteristics in the transition zone are quite extreme, and probably unrealistic. The area outside the transition zone is only slightly more realistic; the whole of the zoomed-in grid is probably not very realistic. We believe that most real distributed programs will fall into the lower left region of the full grid. Accordingly, it is probably safe to ignore the transition zone results, and to view the non-transition zone results as lower bounds on the success rates that can be achieved for real programs. If this is true, then we will be able to select the best 3x2 loner configuration in at least 67% of runs in the full grid, and more likely in more than 67%.

The differences between the transition zone success rate and the non-transition zone success rate vary considerably from one variable to another. Least-commtime changes only seven percentage points, most-nexec changes 14 percentage points, and most-cputime changes the most, 19 percentage points. It is not clear why the sizes of the differences should be so different. In particular, one would expect most-cputime and most-nexec to behave similarly, as they are highly correlated.

The most successful variable/directions are most-nexec, least-commtime, and most-cputime; their success rates are roughly equal (most-nexec is slightly more successful, but it is not clear that we can measure nexec in real programs). The success rate of most-cputime improves the most between the transition zone and the non-transition zone, which may indicate that, beyond the zoomed-in grid, most-cputime becomes more successful than the others.

7-3.3.3.4. Inverse Matching in the Transition Zone

Figure 7-25 shows the best-match success rates of the inverses of some of the more successful variable/directions; the *inverse* of a variable/direction uses the same variable but the opposite direction.

Inverse Best-Matches				
Variable/Direction	Inside Trans. Zone	Set V as a Whole	Outside Trans. Zone	Diff. between Inside and Outside Trans. Zone
least-cputime, TOGETHER	22%	15%	11%	11%
most-commtime, TOGETHER	18%	13%	8%	10%
most-commtime, APART	25%	16%	10%	15%

Figure 7-25:
Inverse best-matches using eql
inside and outside the transition zone of set V
and in set V as a whole

These results are somewhat distressing. Although about 64% of the time the process with the most cputime is the best loner, some 10-20% of the time the process with the least cputime is the best loner. However, when the process with the least cputime is the best loner, it is not necessarily true that the process with the most cputime is the worst loner. This topic is explored further in Section 7-3.3.5.

The success rates of the inverse variable/directions are higher inside the transition zone and lower outside it; the differences range from 10 to 15 percentage points. If the success rates of the inverses continue to fall at such a rate, they will be quite small in the full grid outside the zoomed-in grid, and they will be very small in the lower left of the full grid, the region into which we expect most real programs to fall.

The success rates of least-commtime and most-cputime are about equal outside the transition zone, but the success rate of most-cputime improves more in moving out of the transition zone; if the improvement continues, most-cputime will be more successful than least-commtime in the full grid. For the inverses, the success rates of most-commtime and least-cputime (using TOGETHER) decrease by approximately equal amounts in moving out of the transition zone, but most-commtime is lower in the transition zone, so least-cputime can be expected to be less successful, as desired, in the full grid.

The all-match success rates of the inverse variable/directions are low, as shown in Figure 7-26.

Inverse All-Matches			
Variable/Direction	Inside Transition Zone	Set V as a Whole	Outside Transition Zone
most-commtime, TOGETHER	8%	6%	5%
least-cputime, TOGETHER	8%	8%	7%

Figure 7-26:
Inverse all-matches using eql and TOGETHER
in the transition and non-transition zones of set V
and in set V as a whole

7-3.3.4. Matching Based On Slices

In the preceding section we saw that success rates inside the transition zone are lower than success rates outside it. This might be due to a significant difference between the two zones, or it might be that success rates are lower in the upper left of the grid and rise toward the lower right. In the latter case, any lower right region in the grid would have a higher success rate than any upper left region. To see whether this is the case, we have divided the zoomed-in grid into horizontal, vertical, and diagonal *slices* and compared the matching success rates of the slices.

The horizontal slices are the rows of the grid, which are also CPU isobars; the vertical slices are the columns of the grid. For diagonal slices, we have used some of the COMM isobars, which are curves running generally from lower left to upper right. (Section 4-8 describes isobars in the grids.)

We have done the slicing analyses using the eql TOGETHER matching results; matching using the eql approach and data from the TOGETHER simulations nearly always has the highest success rates. For each type of slicing, we examined the highest all-match and best-match success rates in each slice (row, column, or COMM isobar) and the all-match and best-match success rates of least-commtime in each slice.

There are six cases (cells) in each row and each column of the zoomed-in grid (see Figure 7-11), and there are four traces in each cell, for a total of 24 runs in each row and each column. Few cases in set V fall exactly on a COMM isobar, so we use cases which are close to the isobars. Of the 36 cases in set V, 34 are used in the isobar analysis (all but **vaa** and **vbj**), with from two to six cases (eight to 24 runs) per isobar.

There are no patterns discernible in the row or column slicings, either in the success rates of the most successful variable/directions or of least-commtime. There is a very faint pattern in the COMM isobar slices, in that results are better for the isobars less than or equal to 0.97; for 0.98 and above the results are poorer (see Figure 4-9).

7-3.3.5. Worst Matching

7-3.3.5.1. Introduction

In deciding whether to use matching to select a loner process, and hence a 3x2 configuration, it is important to know the risk involved. One way to define risk is as the percentage of runs in which the configuration selected is not the best loner configuration, but is the middle or worst loner. This is the complement of the success rate, so it can be computed from the success rates given above: the highest best-match success rate is about 64%, so a configuration other than the best is selected in about 36% of our experimental runs. A second way to define the risk is as the frequency of selecting the worst loner. It is this frequency we explore in this section. Keep in mind that these success rates are not probabilities, as in random variables, they are percentages of runs in the zoomed-in grid.

We have noted above that the inverses of the best variable/directions, most-commtime and least-cputime, have success rates as high as 15%. We fear this means that least-commtime will pick the worst loner configuration 15% of the time, but it is not necessarily so. If the best choice has the most commtime, then the worst choice could have either the middle amount of commtime or the least commtime. We want to find out how often least-commtime picks the worst choice and how often it picks the middle choice. Based on the success rates of the inverse variables, we expect to find that least-commtime

picks the best loner	in about 65% of runs,
picks the middle-best loner	in about 20% of runs, and
picks the worst loner	in about 15% of runs.

We further expect to find that most-cputime has similar, though slightly less successful, results.

The success rates of best-oriented matching, discussed previously, tell us how often each variable/direction chooses the best loner configuration. In this section, we seek to discover how often particular variable/directions choose the worst loner configuration. From these figures was can compute how often the middle-best loner is chosen.

To discover how often particular variable/directions choose the worst loner configuration, we introduce *worst-oriented matching*, or just *worst matching* for short. Just as in regular, best-oriented matching we have best-matches and all-matches, in worst matching we have *worst-matches* and *worst-all-matches*, with similar, though reversed, meanings.

The following section, Section 7-3.3.5.2, more fully describes the worst matching procedures and terminology. Section 7-3.3.5.3 presents the worst matching results and conclusions.

7-3.3.5.2. Procedure

Worst matching is almost the same as the regular, best-oriented matching discussed previously, only the senses of the comparisons are reversed. For example, in checking for a most-is-best best-match, regular matching checks whether the process which is the best loner is the same process which has the most of the variable; in checking for a worst-match, the check is whether the process which is the *worst* loner is the same process which has the most of the variable. Most-is-worst worst-all-matching checks whether the worst loner process has the most of the variable, the middle-best loner process has the middle amount of the variable, and the best loner process has the least of the variable. Least-is-best matching is reversed similarly.

Regular matching is not entirely identical to worst matching in the opposite direction. All-match and worst-all-match are, that is, a regular most-is-best all-match is identical to a least-is-worst worst-all-match, and a regular least-is-best all-match is identical to a most-is-worst worst-all-match; see Figure 7-27. However, best-match and worst-match are not the same. A regular most-is-best best-match means that the process which is the best loner has the most of the variable; the worst loner may have the middle amount or the least of the variable. In contrast, a least-is-worst worst-match means that the process which is the worst loner has the least of the variable; the best loner may have the middle amount or the most of the variable; Figure 7-27 illustrates this.

	Regular Matching		Worst Matching	
	most-is-best	least-is-best	most-is-worst	least-is-worst
best	**most**	**least**	least	most
mid	mid	mid	mid	mid
worst	least	most	**most**	**least**

Figure 7-27:
Contrasting Regular Matching and Worst Matching

All three items in a column must match for an all-match or worst-all-match, only the items in bold type need to match for a best-match or worst-match.

We do worst matching using data from the TOGETHER simulations and using the eql approach. The eql TOGETHER combination nearly always has the best results in the best-oriented matching.

7-3.3.5.3. Results and Conclusions

In Figure 7-28, some of the most successful worst-matching results are shown.

Worst-All-Matches			
Variable/Direction	Inside Transition Zone	Set V as a Whole	Outside Transition Zone
least-commtime	8.33%	6.25%	4.76%
most-cputime	8.33%	7.64%	7.14%

Worst-Matches			
Variable/Direction	Inside Transition Zone	Set V as a Whole	Outside Transition Zone
least-commtime	18.33%	15.28%	13.10%
most-cputime	23.33%	17.36%	13.10%

Figure 7-28:
Worst Matching Results in Different Regions

The success rates of some other variable/directions that are fairly successful inside the transition zone are shown in Figure 7-29. The percentages are out of the 60 runs in the set V transition zone.

Inside the Transition Zone of Set V		
Variable/Direction	Worst-All-Match	Worst-Match
most-nio	8.33%	23.33%
most-sumio	6.67%	23.33%
most-iotime	10.00%	18.33%

Figure 7-29:
Worst Matching Success Rates
of some Successful Variable/Directions

As expected, the process selected by the matching procedure as the best loner is in fact the worst loner in about 15-17% of runs in set V when the best two variable/directions are used. Since matching correctly selects the best loner about 65% of the time, the process selected as the best loner is actually the middle-best loner about 20% of the time.

The rate at which bests are picked is lower than we would desire for a placement procedure, and the rate at which mids and worsts are picked is higher. Nevertheless, matching is much more successful than randomly choosing a loner configuration. Random choice should select the best loner about 33% of the time, the middle-best loner about 33% of the time, and worst loner about 33% of the time. Matching does about twice as well as this, choosing the best loner in about 65% of runs.

For the two best variable/directions, least-commtime and most-cputime, the transition zone is ‘‘harder’’ in that the worst loner is selected more often inside the transition zone than outside it. The transition zone was ‘‘harder’’ for regular matching, too, in that the best loner was selected correctly less often in the transition zone than in the non-transition zone.

In summary, matching using least-commtime or most-cputime using the eql approach and data from running (or simulating) the program in the TOGETHER configuration correctly selects the best loner process about 65% of the time, selects as the best loner the process which is actually the middle-best loner about 20% of the time, and selects the process which is actually the worst loner about 15% of the time. Matching is more successful outside the transition zone than inside; this may mean that it is more successful outside the zoomed-in grid.

7-3.3.6. Matching Using Two Variables

7-3.3.6.1. Introduction

All of the matching procedures described so far have used a single variable/direction; our general plan has been to try many approaches and many variable/directions with the objective of finding out which gives the best results. Would using two variable/directions, bringing more information into the placement procedure, result in a higher success rate? That question is the subject of this section.

We believe that using two variable/directions cannot be more successful overall than using any single variable/direction. When a particular process is selected as the best loner process using one of the variable/directions, the other variable/direction must either confirm that selection or disagree. If it agrees, the result is no more successful than using only the first variable/direction alone. If it disagrees, then it is not clear what to do; the process selected using the first variable/direction might be the best one, or the process selected using the second variable/direction might be best, or neither of them might be best; how would the placement procedure decide? So it is not clear how using two variable/directions could improve the success rate. It is, however, probably true that where both variable/directions select the same process, there will be a high success rate.

In the following sections, we analyze and discuss matching using two variable/directions. The objective is to see whether the objections just raised are valid, and to see how often both variable/directions predict the same process. In Section 7-3.3.6.2, we describe the two-variable matching procedure; in Section 7-3.3.6.3, we present the results of doing two-variable matching; and in the final section, Section 7-3.3.6.4, we discuss the results and present our conclusions.

7-3.3.6.2. Procedure

We use only the two variable/directions which were most successful individually: least-commtime and most-cputime. It is hard to believe that any other combination would yield better results. We realize that this is not a complete analysis, and that some other combination might have unexpectedly good results, but the combination analyzed will give a good indication of the potential success of using two variable/directions.

When using two variables to do matching, there are five possible outcomes, which fall into two categories:

I - the same process has most-cputime and least-commtime (‘‘same’’), and
    1 - that process is the best loner (‘‘sameBest’’),
    2 - that process is not the best loner (‘‘sameNotBest’’),

II - different processes have most-cputime and least-commtime (‘‘diff’’), and
    3 - the process with most-cputime is the best loner (‘‘diffCpuBest’’),
    4 - the process with least-commtime is the best loner (‘‘diffCommBest’’),
    5 - neither process is the best loner (‘‘diffNeithBest’’).

As above, ‘‘cputime’’ is the actual elapsed CPU time, and ‘‘commtime’’ is the actual communication waiting time. The eql approach is used, but it happens that no two or three processes have the same values of cputime or commtime. As in other matching work, we use the TOGETHER and APART configurations only.

7-3.3.6.3. Results

The results of matching with most-cputime and least-commtime, using APART and TOGETHER data, are shown in Figure 7-30.

APART						
Category	Category Count	% of 144	Outcome	Outcome Count	% of 144	% of 123 or % of 21
same	123	85.42				
			sameBest	74	51.39	60.16
			sameNotBest	49	34.03	39.84
diff	21	14.58				
			diffCpuBest	3	2.08	14.29
			diffCommBest	11	7.64	52.38
			diffNeithBest	7	4.86	33.33

TOGETHER						
Category	Category Count	% of 144	Outcome	Outcome Count	% of 144	% of 123 or % of 21
same	123	85.42				
			sameBest	81	56.25	65.85
			sameNotBest	42	29.17	34.15
diff	21	14.58				
			diffCpuBest	4	2.78	19.05
			diffCommBest	11	7.64	52.38
			diffNeithBest	6	4.17	28.57

Figure 7-30:
Results of Two-Variable/Direction Matching
using Most-Cputime and Least-Commtime

As with one variable/direction matching, the success rates using data from the TOGETHER configuration are higher than those using data from APART. In this case, the number of sameBest outcomes is higher and the number of diffNeithBest outcomes is lower. In what follows we speak only of the TOGETHER results.

The same process is selected by both variable/directions quite frequently, in 85.42% of the runs. When this happens, the process selected is the best loner in 65.85% of runs. Overall, both variable/directions select the same process, and that process is the best loner (sameBest), in 56.25% of the runs. In 29.17% of the runs the two variable/directions indicate the same process, but it is not the best loner (sameNotBest).

When the two variable/directions select different loner processes (in 14.58% of the runs), least-commtime selects the actual best loner (diffCommBest) in 52.38% of the runs; this is 7.64% of the runs overall. In only 4.17% of the runs does neither variable/direction select the actual best loner process.

7-3.3.6.4. Discussion and Conclusions

The following procedure would seem to be the best way of using two variable/direction matching to select a loner process, and hence a 3x2 configuration:

(1) See which process is selected by each of the two variable/directions, using the TOGETHER data.

(2) If both variable/directions indicate the same loner process, use that process. (This happens in about 85% of the runs in set V. In 66% of those runs the loner so selected was actually the best

loner. Overall, in 56% of the runs, the best configuration was selected this way.)

(3) If the variable/directions select different loner processes, use the one selected by least-commtime. (This happens in about 15% of the runs in set V. In 52% of those runs the loner so selected was actually the best configuration. Overall, in 8% of runs, the best configuration was selected this way.)

This procedure will select the best configuration in 56% + 8% = 64% of the runs. In 36% of the runs it will not select the best configuration; in 29% the variable/directions will select the same configuration, but it will be the wrong configuration, and in 3% + 4% = 7% the variable/directions will select different configurations and least-commtime will select the wrong one.

This procedure has the same 64% success rate in set V as single variable/direction matching using least-commtime. In fact, this two-variable/direction procedure is logically identical to selecting a configuration using only least-commtime: in step (2) we use the configuration selected by least-commtime and also selected by the other variable/direction; in step (3) we use the configuration selected by least-commtime when the other variable/direction selects a different configuration. Clearly it would be preferable to select a configuration in one step, using least-commtime. We conclude that there is no benefit to doing matching using two variable/directions, and that one-variable matching appears to be simpler and faster than two-variable matching.

7-3.3.7. General Conclusions About Matching

Doing matching on least-commtime allows us to successfully predict the full loner order in about 48% of the runs in set V; to be able to predict the full loner order in close to 50% of the runs is quite good. A placement procedure selecting the best loner based on matching using least-commtime would successfully choose the best 3x2 loner configuration in about 64% of the runs in set V. Random choice should select the best loner about 33% of the time, so matching is about twice as good as random choice in set V.

Matching using data from TOGETHER is more successful than matching using data from APART, both in terms of better success rates and lower inverse success rates.

Being successful in 64% of the runs in set V is not the same as being successful 64% of the time. Real distributed programs are probably not uniformly distributed over the zoomed-in grid of set V, indeed they may not fall into the zoomed-in grid at all. On the other hand, we have not examined the success rates of individual variable/directions in each grid cell, so we cannot be certain that it is grid position which determines the success or failure of matching.

A placement procedure based on least-commtime matching would pick the middle-best loner in about 20% of the runs in set V, and the worst loner configuration in about 15% of the runs in set V. So even when matching misses choosing the best loner, it still performs better than a would a random choice between the other two processes.

Matching performs slightly better outside the transition zone than in it, the success rate outside the transition zone rising to about 67% of the runs. This gives us hope that the success rate will continue to rise further away from the transition zone and closer to the lower left region into which we believe real, scientific distributed programs will fall. On the other hand, none of the the slicing analyses showed a pattern of continuing improvement. We will explore this further in the section on matching in full-grid set U (Section 7-3.6.1).

Finally, matching with two variables gives no useful advantage over using only one variable.

This concludes our exploration of the matching technique in zoomed-in-grid set V. We will explore matching in full-grid set U in Section 7-3.6.1, below. For now, we turn our attention to another technique, that of using attractive and repulsive forces to select a 3x2 loner configuration.

7-3.4. Attractive and Repulsive Forces

7-3.4.1. Introduction

Another approach to choosing the best loner configuration in a 3x2 situation is to look at the ‘‘forces’’ which ‘‘attract’’ processes toward each another and ‘‘repel’’ them away from one another. Two processes which communicate frequently or voluminously may benefit from being placed together on the same processor; we can say that they are ‘‘attracted’’ to each other, and that there is an attractive force between them, like magnetism, whose strength is proportional to the amount of communication between them. Similarly, if two processes both have heavy CPU demands, they may benefit from being placed on separate processors; we can say that they ‘‘repel’’ each other, and that there is a repelling force between them, again like magnetism, whose strength is proportional to the amount of CPU time consumed. There may be other attracting or repelling forces besides communication activity and CPU use.

In a 3x2 situation, we could choose a loner process by looking at a certain attractive force to see which two processes are most strongly attracted to each other. Those two processes could then be placed together, and the remaining process would be the loner. Similarly, we could look at repelling forces to choose a loner: the two strongest repelling forces would determine the loner process. Of course, we could use various combinations of attracting and repelling forces. In the latter case, different ‘‘forces’’ may be measured in different units, and we may have a problem finding a way to convert to comparable units.

In general, in an NxM situation, with N processes and M≤N processors, we could use the attractive or repulsive forces to cluster the processes into subsets, which could be placed on different processors. This would probably involve an iterative procedure of finding the two or more most attracted (or least repelled) processes, coalescing them and combining their characteristics, and then repeating, until only M clusters remain.

We have tried to find the strength of various attractive and repulsive forces, to use them in a variety of ways to make predictions as to the best loners, and to test those predictions against the actual best loners. We describe these procedures and their results in the following sections.

7-3.4.2. Description

The attractive forces we have investigated are all variations of the amount of communication between two processes. There are, however, several ways the amount of communication might be measured, and so, several ways to gauge the strength of the attractive force, one of which might be more effective at finding the best loner than another. Specifically, we have used three measurements of the attractive force of communication: (1) the number of messages sent between each pair of processes, (2) the total size (in bytes) of the messages sent between each pair of processes, and (3) the total transmission time of the messages sent between each pair of processes. We have used one measure of the repulsive force: the sum of the elapsed CPU time for each pair of processes. We have used one measure which combines attractive and repulsive forces: the combined transmission time and CPU time of each pair of processes. These measures are summarized in Figure 7-31 and explained in greater detail in the following paragraphs.

Type of Force	What is Measured	Measure Used
Attractive	Communication	Message Count Sum of Message Sizes Sum of Transmission Times
Repulsive	CPU Use	Sum of Elapsed CPU Time
Combination	Comm. and CPU Use	Trans. Time–CPU Use

Figure 7-31:
Attractive and Repelling Forces and Measures Used

7-3.4.2.1. Attractive Forces

The message count and the sum of message sizes for each pair of processes are gathered from the trace summary for each run.

In Carnival, local messages have zero transmission time; send and receive CPU time are all that is necessary to deliver the message. Remote messages have a latency delay, which is the same for all messages, plus a delay due to transmission, which is proportional to message size. For this investigation, remote delay is computed the same way Carnival computes it. What we really want to use as a measure of attractive force is the difference between remote and local transmission times; since the local transmission time is zero, the difference is the same as the remote transmission time.

In Carnival, the calculation of transmission time uses the parameters *comm_latency* and *comm_bandwidth*. *Comm_latency* is in milliseconds. The *comm_bandwidth* parameter is given in megabits per second (Mb/s), which is converted internally into the variable *cbw* (communication bandwidth) which is in bytes/ms. The transmission time of a message, or the time the message spends passing through the network, is

$$transmission\_time = comm\_latency + (message\_size / cbw)$$

the units being

$$ms = ms + (bytes\ /\ (bytes/ms)).$$

In Carnival, the transmission time of each message is calculated as the message is sent. For this investigation we calculated the total transmission time of all messages sent between two processes, *p1* and *p2*, as

$$total\_transmission\_time\,[p1,\ p2]$$
$$= (msg\_count\,[p1,\ p2] \cdot comm\_latency) + (sum\_msg\_sizes\,[p1,\ p2]/cbw)$$

In all the Carnival simulations, *comm_latency* has been set to 1.5 ms and *comm_bandwidth* to 100 Mb/s.†

To use the attractive forces, we find the two processes between which the force is strongest. Since those two processes are most strongly attracted, we predict that the best loner must be the third, remaining, process. For example, if

---

† Actually, the Carnival parameter files contain the value 10 Mb/s for *comm_bandwidth*, but due to a bug in early versions of Carnival, this is used internally as though it were 100 Mb/s. For this investigation, we have used the same values of 1.5 and 10, and we have replicated the internal bug from Carnival, so that the values are effectively 1.5 and 100, as in Carnival.

$$\begin{aligned} msg\_count[0,1] &= 318 \text{ messages} \\ msg\_count[1,2] &= 277 \text{ messages} \\ msg\_count[2,0] &= 146 \text{ messages} \end{aligned}$$

then there are more messages sent between processes 0 and 1 than between any other pair of processes. Processes 0 and 1 are most strongly attracted and should be placed together, leaving process 2 to be the loner.

7-3.4.2.2. Repulsive Force

The elapsed CPU time used by each process is gathered from the simulation outputs. The CPU service demanded by a process varies somewhat depending on which of the five configurations is used because the sending of local and remote messages require different amounts of CPU service, and the elapsed time it takes for the process to receive the CPU service it demands varies greatly depending on which other processes are sharing the CPU. Matching was most successful using data from the TOGETHER configuration, so, to make these results comparable, we used the elapsed CPU times only from the TOGETHER configuration.

To use the repulsive force, we must find which process is most strongly repelled by all other processes. For example, if

$$\begin{aligned} cpu\_sum[0,1] &= 45{,}604 \text{ ms} \\ cpu\_sum[1,2] &= 32{,}119 \text{ ms} \\ cpu\_sum[2,0] &= 63{,}985 \text{ ms} \end{aligned}$$

then the sum of the CPU times of processes 1 and 2 is smaller than than that of any other pair of processes; processes 1 and 2 are most weakly repelled, thus process 0 is most strongly repelled by the other processes and should be the loner.

7-3.4.2.3. Combination of Attractive and Repulsive Forces

The larger the total transmission time of two processes, the more strongly they are attracted together; the larger the sum of the CPU times of two processes, the more strongly they are repelled apart. If we subtract the sum of the CPU times from the transmission time for each pair of processes, the largest remainder indicates the strongest remaining attraction. The two most strongly attracted processes should be placed together, and the remaining process should be the best loner. Since this approach takes into account both attraction and repulsion, both CPU activity and communication activity, we expected it to be more successful at predicting the best loner than either of the separate approaches.

We compute the transmission-time/CPU-time difference for each pair of processes, *p1* and *p2*, using the transmission time and the sum of elapsed CPU time previously calculated:

$$\begin{aligned} &transmission\_cpu\_diff[p1,\ p2] \\ &\quad = transmission\_time[p1,\ p2] - cpu\_sum[p1,\ p2] \end{aligned}$$

There is no problem with units of measure in this combination, since transmission time and elapsed CPU time are both in milliseconds.

7-3.4.3. Procedure

For each run we make best-loner predictions according to each of the attractive, repulsive, and combination approaches. We also make a prediction according to which single process has the largest elapsed

CPU time, predicting that the process with the largest CPU time will be the best loner; this is the same as matching on most-cputime. Each prediction is compared to the actual loner response times; the result for each prediction is ‘‘best,’’ ‘‘mid,’’ or ‘‘worst,’’ depending on whether the process selected as best loner is actually the best loner, middle-best loner, or worst loner, respectively.

7-3.4.4. Results on the Grid

In order to see if there is any pattern in the grid, we counted the number of best results for each case, out of the four repetitions of each case, for each of the five measures. As an example, the grid for the combined forces approach is shown in Figure 7-32.

	IO-COMM					
CPU	IO 0 COMM 100	IO 1 COMM 99	IO 2 COMM 98	IO 3 COMM 97	IO 4 COMM 96	IO 5 COMM 95
0	**vaa** 1	**vab** 3	**vac** 3	**vad** 3	**vae** 2	**vaf** 2
1	**vag** 2	**vah** 2	**vai** 4	**vaj** 2	**vak** 2	**val** 3
2	**vam** 2	**van** 2	**vao** 3	**vap** 3	**vaq** 1	**var** 3
3	**vas** 3	**vat** 4	**vau** 2	**vav** 1	**vaw** 3	**vax** 2
4	**vay** 1	**vaz** 4	**vba** 4	**vbb** 1	**vbc** 3	**vbd** 3
5	**vbe** 1	**vbf** 3	**vbg** 3	**vbh** 3	**vbi** 2	**vbj** 4

Figure 7-32:
Number of Best Results in each Cell
for Combined Forces Approach

There do not appear to be any patterns in the distribution of large numbers of best results in any of the grids we examined. There does not seem to be any reason to expect that the combined approach would be either more or less successful in other regions of the full grid than it is in the zoomed-in grid of set V.

7-3.4.5. Results for the Set

To get the overall results for set V, we count the number of times each approach had each result, over all runs in the set. The results are shown in Figure 7-33.

Force(s)	Approach	Result	Count	% of 144
Attractive	Message Count	best	46	31.94%
		mid	46	31.94%
		worst	52	36.11%
	Sum of Message Sizes	best	40	27.78%
		mid	47	32.64%
		worst	57	39.58%
	Transmission Time	best	46	31.94%
		mid	47	32.64%
		worst	51	35.42%
Repulsive	Pairwise CPU Time	best	85	59.03%
		mid	34	23.61%
		worst	25	17.36%
Combination	Trans. Time and CPU Time	best	90	62.50%
		mid	37	25.69%
		worst	17	11.81%
Matching on most-cputime	Single Largest CPU Time	best	85	59.03%
		mid	34	23.61%
		worst	25	17.36%

Figure 7-33:
Success of Attractive and Repulsive Forces in Set V

7-3.4.6. Conclusions

The three attractive-force approaches each select the best, middle, and worst loner process about one-third of the time. In other words, these approaches do no better than randomly choosing a loner process. The fact that the three approaches have very similar results suggests that it really does not matter how we measure the attractive force.

The repulsive-force approach using sum of CPU time is more successful than the attractive-force approaches: almost 60% bests, another 24% mids, and only about 17% worsts. The repulsive-force approach has exactly the same success as matching most-cputime, and on further investigation they turn out to be functionally the same.

Combined attraction and repulsion, the approach we expected to have the best success, does indeed have the best success of any of these approaches, with 62.5% bests, about 26% mids, and only about 12% worsts. This 62.5% success rate is certainly much better than the 33% success rate that can be expected from randomly choosing a loner process, but it is not as high as we would like.

7-3.5. Conclusions About Set V as a 3x2: Finding the Best Loner

The best result for the matching technique in set V is about 64% successes; this is achieved by selecting as the loner process the process with the least communication time. The best result for the attractive/repulsive forces technique in set V is about 62.5% successes; this is achieved by using both message transmission time and CPU time to select the loner process. These success rates are very close; the difference is probably not significant, so we cannot choose between the two techniques based on the success rates in set V. The matching technique would be somewhat less expensive to use, in that it requires only one type of data, whereas attractive/repulsive forces requires two.

7-3.6. Set U as 3x2

Experiment set V, examined in the preceding sections, is based on the zoomed-in grid, the region where it is hardest to select the best loner configuration. We expect that very few real distributed programs will fall into the zoomed-in grid; most programs will fall into the full grid outside the zoomed-in grid region. To see whether the two techniques described above, matching and attractive/repulsive forces, are successful at selecting the best loner configuration in more realistic 3x2 situations, we applied them in full-grid experiment set U. Having explored both techniques fairly extensively in set V, we desire only to get a general idea of the success rates of the techniques in set U.

Section 7-3.6.1 describes matching in set U, followed by Section 7-3.6.2 on attractive/repulsive forces in set U. Section 7-3.6.3 contains our conclusions about the 3x2 situation in set U.

7-3.6.1. Matching in Set U

Our objective is only to find the success rates of matching in set U; we do not repeat the extensive analyses done in set V. We present all-match and best-match results, using data from TOGETHER and APART simulations; we present only the eql results, not the noeql results. There are 26 cases in set U, with two repetitions of each case, for a total of 52 runs in the set. The results of matching in set U are shown in Figure 7-34, including the most successful variable/directions and two inverse variable/directions.

All-Match				
Variable	TOGETHER		APART	
	Count	% of 52	Count	% of 52
most-cputime	31	59.62%	30	57.69%
most-sumexec	24	46.15%	24	46.15%
least-commtime	23	44.23%	20	38.46%
most-naction	23	44.23%	23	44.23%
most-nexec	20	38.46%	20	38.46%
most-commtime	1	1.92%	0	0%
least-cputime	0	0%	0	0%

Best-Match				
Variable	TOGETHER		APART	
	Count	% of 52	Count	% of 52
most-cputime	40	76.92%	43	82.69%
most-sumexec	33	63.46%	33	63.46%
most-naction	33	63.46%	33	63.46%
most-nexec	30	57.69%	30	57.69%
least-commtime	29	55.77%	29	55.77%
most-commtime	2	3.85%	3	5.77%
least-cputime	0	0%	1	1.92%

Figure 7-34:
Eql Matches in set U
The Most Successful Variable/Directions and Two Inverse Variable/Directions

Most of the success rates are much higher than in set V. In set V, the highest best-match success rate was approximately 64%, in set U the best-match success rates go up to about 83%.

In set V, the TOGETHER data was nearly always more successful. Here, TOGETHER is usually better, though sometimes APART is the same or very close. Best-match on most-cputime is an exception in which APART is better.

Most-cputime is clearly the most successful variable/direction for both best-match and all-match. In set V, most-cputime and least-commtime were about equally successful. In set U, least-commtime is less successful than it was in set V, while most-cputime is much more successful than it was in set V (see Figures 7-22 and 7-23). In Section 7-3.3.3.3, we noted that the rate of improvement as one moved down and right in set V was greater in most-cputime than in least-commtime. This apparently is the outcome.

The 83% rate of success is somewhat puzzling. Although matching was more successful outside the transition zone than inside it, the slicing analysis did not find a continuing improvement as one moved outward from the upper left corner. We did not expect much better results beyond the set V zoomed-in grid.

7-3.6.2. Attractive and Repulsive Forces in Set U

We repeated the attractive and repulsive forces investigation for set U. The results are shown in Figure 7-35.

Force(s)	Approach	Result	Count	% of 52
Attractive	Message Count	best	12	23.08%
		mid	14	26.92%
		worst	26	50.00%
	Sum of Message Sizes	best	12	23.08%
		mid	14	26.92%
		worst	26	50.00%
	Transmission Time	best	12	23.08%
		mid	14	26.92%
		worst	26	50.00%
Repulsive	Pairwise CPU Time	best	40	76.92%
		mid	12	23.08%
		worst	0	0%
Combination	Trans. Time and CPU Time	best	40	76.92%
		mid	12	23.08%
		worst	0	0%
Matching on most-cputime	Single Largest CPU Time	best	40	76.92%
		mid	12	23.08%
		worst	0	0%

Figure 7-35:
Success of Attractive and Repulsive Forces in Set U

As in set V, the three attractive-force approaches all have the same results, and matching with most-cputime has the same result as the repulsive-force approach.

The success rates of the attractive-force approaches are much lower in set U than in set V. With only 23% bests and 50% worsts we must discard all the attractive force approaches immediately. They are worse than randomly selecting a loner configuration.

The success rate of the repulsive-force approach is substantially better than in set V (see Figure 7-33), reaching a fairly good 77% bests. Unlike in set V, the combined attractive-repulsive approach has the same success rate as the plain repulsive-force approach. The pattern of successes in set V did not appear to be better for the non-transition zone than for the transition zone, nor did the pattern of successes appear to be better lower down in the set V grid, so we had no expectation that it would be more successful outside the set V grid. Apparently, in the full grid computation is so much more important than communication that it completely dominates the combined results.

7-3.6.3. Conclusions About Set U as a 3x2: Finding the Best Loner

The best result for the matching technique in set U is about 83% successes using the eql approach; this is achieved by selecting as the loner process the process with the largest CPU demand in the APART simulations. The best result for the attractive/repulsive forces technique in set U is about 77% successes; this can be achieved by using any of the approaches involving the repulsive force of CPU demand, using data from the TOGETHER simulations. The eql approach may be overstating the success of matching, but it appears that matching using APART data is at least as good as or somewhat better than attractive/repulsive using TOGETHER data.

Attractive/repulsive was run only with the TOGETHER data because in set V the TOGETHER data was more successful. With the TOGETHER data, the repulsive and combined approaches have the same results, and they are the same results as matching on most-cputime. The repulsive pairwise CPU approach is functionally identical with matching on most-cputime, so it seems very likely that the repulsive and combined results using APART data would be the same 83% as matching on most-cputime using APART data. Based on this supposition, we can say that, in set U, the best matching results and the best attractive/repulsive results are probably about the same. The matching technique or the repulsive forces technique would be somewhat less expensive to use, in that either of them requires only one type of data, whereas the combined attractive and repulsive forces technique requires two.

7-3.7. Conclusions About Sets U and V as 3x2: Finding the Best Loner

In both set U and set V, the matching and the attractive/repulsive forces techniques have approximately the same results, about 62-64% best-matches in set V and about 83% best-matches in set U. In zoomed-in set V, matching using TOGETHER has better results than matching using APART, and the best matching results come from least-commtime, with most-cputime second. In full-grid set U, the best matching results come from most-cputime using APART data; matching using least-commtime is much worse. In set V, the best attractive/repulsive force is the combined approach using both CPU and communication information. In set U, the combined approach has the same results as the repulsive-only, CPU-only approach.

It appears that, over the full grid, CPU time is the most important factor in selecting the best configuration. In the zoomed-in grid, communication increases in importance, but CPU time is still more important.

7-4. Regions of the Grid

7-4.1. Introduction

The full grid, used in experiment sets N and U, covers all possible distributed programs; the zoomed-in grid, used in sets R and V, covers the upper left corner of the full grid in more detail. In this section, we explore some other regions of the full grid.

7-4.1.1. The Lower Left Regions

Real distributed programs are either distributed approximately evenly across the full grid, or they are clustered in one or a few regions of the grid; we believe that they probably cluster in some region of the grid. We know how well our techniques work in each part of the grid; if we knew where real programs fall, we could tell how well our techniques would work for real programs.

We can apply a bit of deduction and figure out where in the grid real programs are likely to fall. Real distributed programs are probably large and expensive to run, otherwise they would not be worth distributing. In other words, real distributed programs are probably scientific programs. These programs probably have a very high proportion of CPU use, and do little communication or I/O. To the extent that they do either, they probably do more communication than I/O. Most likely a parent process or master process reads in whatever input information is needed, then distributes it to the child or slave worker processes by sending messages. During the computation, the worker processes would have to communicate among themselves, to synchronize, to share intermediate results, and so on. The region thus described is in the lower part of the full grid, where the proportion of CPU execute actions is high, and on the left side, where the ratio of communication to I/O is high.

In Sections 7-4.2 and 7-4.3 we explore the lower left part of the full grid. In Section 7-4.2 we explore two subsets of the set U full grid, and in Section 7-4.3 we create experiment set W, an intermediate grid zooming in on the lower left cell.

7-4.1.2. The Lower Right Region

With both two and three processes, our results are better over the full grid (sets N and U) than in the zoomed-in grid (sets R and V). Since the results are worse than average in some regions of the grid, they should be better than average in some other regions. We would like to know in which regions the results are better than average. We hypothesize that the lower right of the full grid is an above-average region, and in Section 7-4.4 we explore this hypothesis.

7-4.2. The Sci and VSci Regions

7-4.2.1. Introduction

Let us define a region in the lower left part of the set U full grid into which we believe real, scientific distributed programs fall. We call it the *scientific region*, or *sci region;* it is depicted in Figure 7-36.

	IO-COMM				
CPU	IO 0 COMM 100	IO 25 COMM 75	IO 50 COMM 50	IO 75 COMM 25	IO 100 COMM 0
0					
20					
40	**uk**	**ul**			
60	**up**	**uq**			
80	**uu**	**uv**			
100	**uz**				

Figure 7-36:
Scientific (Sci) Region of Set U

The sci region is fairly large; it encompasses a substantial portion of the full grid, and it includes programs with as little as 40% CPU actions. Let us define a smaller region which more strongly represents the characteristics we wish to focus on, and call it the *very scientific region*, or *vsci region.* The vsci region consists of cells **uu**, **uv**, and **uz**.

The scientific region includes seven cases; with two repetitions per case, it includes 14 runs. Fourteen runs is enough to give us some meaningful indications, but it is probably not statistically significant. The very scientific region includes only three cases; with two repetitions per case, it includes only six runs. Six runs is too few to be significant, but we are nonetheless interested in the outcomes in that region.

We must be especially wary of percentages in these regions, since an increase or decrease of one in the number of successful predictions is about a 7% change in the sci region, and about a 17% change in the vsci region.

In the following sections, we discuss the matching and attractive/repulsive forces techniques in the sci and vsci regions.

7-4.2.2. Matching in the Sci and VSci Regions

We carried out the matching analysis in the scientific and very scientific regions. Figure 7-37 shows the best-match results using the eql approach for the two regions.

Sci Region				
Variable/Direction	APART		TOGETHER	
	Count	% of 14	Count	% of 14
least-commtime	9	64.29%	8	57.14%
most-sumexec	11	78.57%	11	78.57%
most-cputime	12	85.71%	10	71.43%

VSci Region				
Variable/Direction	APART		TOGETHER	
	Count	% of 6	Count	% of 6
least-commtime	3	50.00%	3	50.00%
most-sumexec	6	100.00%	6	100.00%
most-cputime	6	100.00%	5	83.33%

Figure 7-37:
Best-matches Using Eql
Most Successful Variable/Directions
in the Sci and Vsci Regions

In the sci region, the best variable/direction is most-cputime, which is successful in 71-86% of runs. Predictions based on the APART simulations get one or two more successful predictions, or 6-12% better, than those based on the TOGETHER simulations.

In the vsci region, the best variable/directions are most-cputime and most-sumexec, which are successful in about 83-100% of runs. Predictions based on the APART simulations are as successful or slightly more successful than those based on the TOGETHER simulations.

The best-match success rates in set V are about 53-63%; in set U they are about 55-83%. The success rates in sci are slightly better, about 2% better, than the results for all of set U.

Because of the large granularity, it is hard to be sure about the results in vsci. With some variable/directions, matching succeeds in five out of six runs, which is 83%, which is about the same as in all of set U and in the sci region. In other cases, it succeeds in six out of six runs, which is 100%; 100% looks much better than 83% but is really only a difference of one successful prediction. It appears that the results in vsci are somewhat better than for sci, which in turn are slightly better than the results for all of set U, which are in their turn quite a bit better than the results for set V.

In the results for set V, the results using the TOGETHER simulations were almost always better than the results using the APART simulations. In set U, APART was very slightly better than TOGETHER, and in the sci and vsci regions APART was again slightly better than TOGETHER.

7-4.2.3. Attractive and Repulsive Forces in the Sci and VSci Regions

We carried out the attractive and repulsive forces analysis in the sci and the vsci regions; the results are shown in Figure 7-38.

Force(s)	Approach	Result	Sci Region		Vsci Region	
			Count	% of 14	Count	% of 6
Attractive	Message Count	best	2	14.29%	1	16.67%
		mid	4	28.57%	3	50.00%
		worst	8	57.14%	2	33.33%
	Sum of Message Sizes	best	2	14.29%	1	16.67%
		mid	4	28.57%	3	50.00%
		worst	8	57.14%	2	33.33%
	Transmission Time	best	2	14.29%	1	16.67%
		mid	4	28.57%	3	50.00%
		worst	8	57.14%	2	33.33%
Repulsive	Pairwise CPU Time	best	10	71.43%	5	83.33%
		mid	4	28.57%	1	16.67%
		worst	0	0%	0	0%
Combination	Trans. Time and CPU Time	best	10	71.43%	5	83.33%
		mid	4	28.57%	1	16.67%
		worst	0	0%	0	0%
Matching on most-cputime	Single Largest CPU Time	best	10	71.43%	5	83.33%
		mid	4	28.57%	1	16.67%
		worst	0	0%	0	0%

Figure 7-38:
Success of Attractive and Repulsive Forces
in the Sci and VSci Regions of Set U

In each region, all the predictions based on attractive forces have exactly the same results, and the predictions based on the combined forces have exactly the same results as the repulsive-forces-only predictions.

The attractive predictions are very poor. In each region only about 14-17% successfully predict the best loner. Close to 60% of the predictions choose the worst loner in sci, and 33% choose the worst loner in vsci.

The repulsive and combined predictions were fairly good: in sci, the best loner was selected in 71% of runs; in vsci, the best loner was selected in 83% of runs.

The success rates in the sci region are slightly worse than the set U results, while the success rates in the vsci region are slightly better than the set U results. Considering the small numbers of cases and the the granularity of the percentages, it seems most likely that the success rates for set U as a whole, the sci region, and the vsci region are about the same.

7-4.2.4. Conclusions About the Sci and VSci Regions

In the sci and vsci regions, the success of the attractive/repulsive forces technique (71% and 83%, respectively) is not as good as the success of the matching technique (79% and 83-100%).

The success rates of both the matching and attractive/repulsive forces techniques are about the same or slightly more successful in the sci and vsci regions than in all of set U. If real distributed programs do actually fall into the sci or vsci regions of the grid, we can expect that the techniques we have explored for dealing with 3x2 situations will have approximately the success rates we found in set U, or slightly better.

7-4.3. Experiment Set W

The scientific (sci) and very scientific (vsci) regions of set U are not sufficient to study the region of the grid into which we believe many or most real distributed programs will fall. The sci region covers a significant part of the full grid. The vsci region, too, covers a large range of cases. At the same time, both sci and vsci include small numbers of cases (14 and six, respectively), making it difficult to draw firm conclusions. Most importantly, real scientific programs are probably close to 100% CPU actions, whereas both sci and vsci cover a considerable range of percentages of CPU actions. Therefore we decided to examine more closely the part of the grid into which we believe real distributed programs fall.

We zoom in on the lower left cell of the full grid, the cell labelled **nz** in set N and **uz** in set U. (Perhaps we should call it the *very very scientific region*.) We created a grid which is an intermediate-scale zoom-in of cell **uz**, on the same scale as set P, with increments of five percentage points both vertically and horizontally. Generating three-process traces with characteristics dictated by this grid creates experiment set W, shown in Figure 7-39.

	IO-COMM					
CPU	IO 0 COMM 100	IO 5 COMM 95	IO 10 COMM 90	IO 15 COMM 85	IO 20 COMM 80	IO 25 COMM 75
80	**wa**	**wb**	**wc**	**wd**	**we**	**wf**
85	**wg**	**wh**	**wi**	**wj**	**wk**	**wl**
90	**wm**	**wn**	**wo**	**wp**	**wq**	**wr**
95	**ws**	**wt**	**wu**	**wv**	**ww**	**wx**
100	**wy**					

Figure 7-39:
Set W

There are 25 cells in set W. We generated two traces per cell, for a total of 50 runs in the set. We simulated all five configurations of each trace. In the following sections we briefly analyze set W in both 3x3 and 3x2 situations.

7-4.3.1. Set W as 3x3

In a 3x3 situation, in which the APART configuration can be used, APART is the best configuration in every run in set W. This confirms our belief that, for nearly all real distributed programs, APART will be the best configuration, thus the always-APART policy should be perfectly successful for such programs.

Dyadic placement predicts that APART will be the best configuration in every run in set W. This confirms earlier results that dyadic placement is accurate except in the top (zero percent CPU) row of the full grid.

7-4.3.2. Set W as 3x2 Analyses

In this section we apply the matching and attractive/repulsive forces techniques in set W and analyze the results. The results in the sci and vsci regions lead us to expect success rates of 85-100% for both techniques in set W.

7-4.3.2.1. Matching in Set W

Figure 7-40 shows the best-match results using the eql approach in set W.

Set W				
Variable/Direction	APART		TOGETHER	
	Count	% of 50	Count	% of 50
least-commtime	39	78.00%	41	82.00%
most-nexec	42	84.00%	42	84.00%
most-sumexec	41	82.00%	41	82.00%
most-cputime	41	82.00%	42	84.00%
most-naction	42	84.00%	42	84.00%

Figure 7-40:
Eql Best-match Success Rates in set W

The success rates using the TOGETHER simulations are slightly better than those using the APART simulations. The success rates are about 84%, which is about the same as the success rates in the sci and vsci regions of set U. As in earlier work, most-cputime is the best or nearly the best variable to use.

7-4.3.2.2. Attractive and Repulsive Forces in Set W

Figure 7-41 shows the results of using the attractive, repulsive, and combined forces approaches on the 50 runs in set W.

Force(s)	Approach	Result	Count	% of 50
Attractive	Message Count	best	15	30.00%
		mid	10	20.00%
		worst	25	50.00%
	Sum of Message Sizes	best	15	30.00%
		mid	10	20.00%
		worst	25	50.00%
	Transmission Time	best	15	30.00%
		mid	10	20.00%
		worst	25	50.00%
Repulsive	Pairwise CPU Time	best	42	84.00%
		mid	8	16.00%
		worst	0	0%
Combination	Trans. Time and CPU Time	best	42	84.00%
		mid	8	16.00%
		worst	0	0%
Matching on most-cputime	Single Largest CPU Time	best	42	84.00%
		mid	8	16.00%
		worst	0	0%

Figure 7-41:
Success of Attractive and Repulsive Forces in Set W

As in earlier work, all the attractive forces have the same success rates, and those results are very poor, worse than random. Also as in previous work, the repulsive force and the combined forces have the same results. The repulsive forces have 84% success, which is a little better than the success rates in the sci region, and about the same as in the vsci region.

As in earlier work, the matching techniques have about the same success rates or slightly better than the attractive/repulsive forces technique, and the matching technique is easier to carry out.

7-4.3.3. Conclusions About Set W

The best success rates for matching and attractive/repulsive forces in set W are about the same as those in all of set U. That is, it is possible to select the best 3x2 loner configuration in about 84% of runs for scientific (heavy CPU, more communication than I/O) distributed computations.

The best success rates for matching in set W are about 84%, which is about the same as the best success rates in the sci and vsci regions of set U. There does not seem to be a trend as we move deeper into the lower left-hand corner, from the sci region, to the vsci region, to set W. We have seen worse-than-average results in the transition zone in the upper left, and we now see average results in the sci and vsci regions of set U and in set W.

7-4.4. The LR Region

7-4.4.1. Introduction

In the upper left corner of the grid (the area covered by sets R and V) it is harder to predict the outcomes, that is, most of our techniques are less successful there than they are over the full grid (covered by sets N and U). If the success rates over the full grid are ‘‘average,’’ and those in the upper left-hand corner region are ‘‘below average,’’ then there must be regions in which they are ‘‘above average.’’ We would like to know if there are such above average regions, and if so, where they are.

We have investigated the ‘‘scientific’’ and ‘‘very scientific’’ regions and set W, all of which are in the lower left-hand corner of the grid, and found that the success rates in that region are close to those for the full grid. (Actually, the success rates in sci and vsci are slightly better, but only slightly.) So the lower left region of the gird is not an above average region.

The transition zones of the zoomed-in grids (sets N and R) are bounded by diagonal lines from middle left to upper right. Above and left of that line the worst success rates occur; below and to the right of it the success rates are higher. Perhaps an above-average region is symmetrically located, as shown in Figure 7-42.

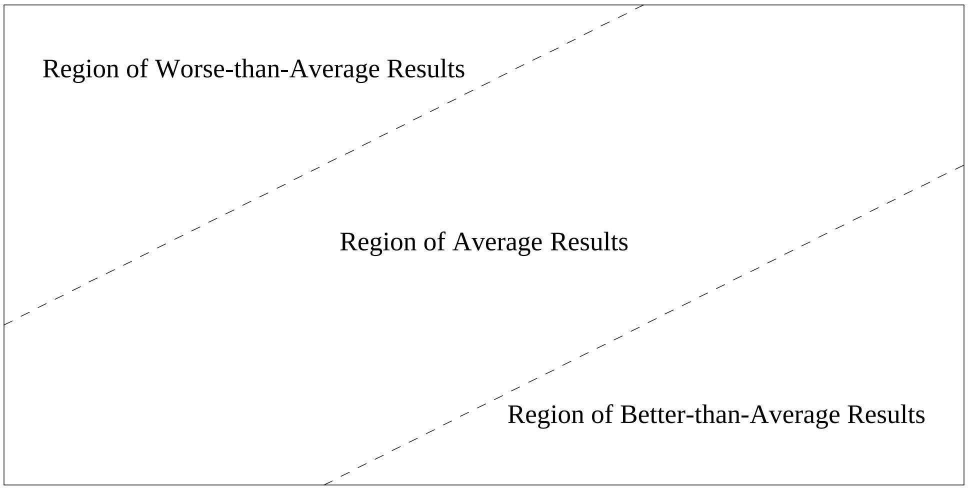
Region of Worse-than-Average Results
Region of Average Results
Region of Better-than-Average Results

Figure 7-42:
Hypothesized Regions of Below Average, Average, and Above Average Results

We define a region in the lower right of the set U full grid, called the *LR region*, (for lower right), as shown in Figure 7-43.

	IO-COMM				
CPU	IO 0 COMM 100	IO 25 COMM 75	IO 50 COMM 50	IO 75 COMM 25	IO 100 COMM 0
0					
20					
40					**uo**
60				**us**	**ut**
80			**uw**	**ux**	**uy**
100					

Figure 7-43:
The Lower Right (LR) Region of set U

The LR region includes six cases; with two repetitions per case, it includes 12 runs. That is a large enough number of runs to give some meaningful indications, but it is probably not statistically significant.

In the following sections, we briefly look at the success of the matching and attractive/repulsive forces techniques in the LR region.

7-4.4.2. Matching in the LR Region

We carried out a matching analysis on the LR region. The best-match results using the eql approach are shown in Figure 7-44.

LR Region				
	TOGETHER		APART	
Variable/Direction	Count	% of 12	Count	% of 12
least-commtime	4	33.33%	4	33.33%
most-sumexec	10	83.33%	10	83.33%
most-cputime	10	83.33%	10	83.33%

Figure 7-44:
Best-matches Using Eql
Most Successful Variable/Directions
in the LR Region

In the LR region, there are 10 successful predictions out of 12, which is about 83%. Predictions based on the APART and TOGETHER simulations are identically successful.

The success rates of various versions of matching in set V are about 53%-63%. The success rates in set U are about 55%-83%. In the sci region they are about 86%, and in vsci they are very approximately 83-100%. In set W, the success rates are about 84%.

The success rates in the LR region are about the same as the best success rates for matching in set U, and about the same as in the lower left regions (the sci and vsci regions and set W). In other words, the success rates of matching the the LR region are average, not above average.

7-4.4.3. Attractive and Repulsive Forces in the LR Region

We carried out an attractive and repulsive forces analysis in the LR region and the results are shown in Figure 7-45.

Force(s)	Approach	Result	Count	% of 12
Attractive	Message Count	best	5	41.67%
		mid	3	25.00%
		worst	4	33.33%
	Sum of Message Sizes	best	5	41.67%
		mid	3	25.00%
		worst	4	33.33%
	Transmission Time	best	5	41.67%
		mid	3	25.00%
		worst	4	33.33%
Repulsive	Pairwise CPU Time	best	10	83.33%
		mid	2	16.67%
		worst	0	0%
Combination	Trans. Time and CPU Time	best	10	83.33%
		mid	2	16.67%
		worst	0	0%
Matching on most-cputime	Single Largest CPU Time	best	10	83.33%
		mid	2	16.67%
		worst	0	0%

Figure 7-45:
Success of Attractive and Repulsive Forces
in the Lower Right Region of Set U

In LR, as in earlier work, all the predictions based on attractive forces have exactly the same success rate, and the predictions based on combined attractive and repulsive forces have exactly the same success rate as the repulsive-forces-only predictions. The attractive predictions have low success rates, as usual. The attractive and combined predictions were quite good: succeeding in 10 of 12 runs, or about 83%.

In set V, the attractive forces were all about 33% successful, the repulsive forces were 59% successful, and the combined measure was 62.5% successful. In set U, the attractive forces were about 23% successful, and the repulsive and combined forces were about 77% successful. In sci the repulsive and combined forces were about 71% successful; in vsci they were about 83% successful; and in set W, about 84% successful.

The success rates in the LR region are about the same as the best success rates in set U, and about the same as in the lower left regions (the sci and vsci regions and set W). In other words, the success rates of matching the the LR region are average, not above average.

7-4.4.4. Conclusions About the LR Region

Neither the matching approach nor the attractive and repulsive forces approach have higher success rates in LR than they do over the full set U grid. It is clear that LR is not an above average region; furthermore, there does not seem to be any support for the hypothesis that there are above average regions in the grid.

Notice, however, that matching in the LR region of set U succeeds in 10 of 12 runs, or 83.3%. Matching in the upper left corner of set U, defined as cells **ua**, **ub**, **uc**, **uf**, **ug**, and **uk**, succeeds in 9 of 12 runs, or 75%. Although this appears to be a sizable 8% difference, it is really only a difference of one in the number of successful matches. We consider the upper left corner to be a below-average region due to poor performance in the transition zone, yet the poor performance is not evident in the full grid, it is only evident in the zoomed-in grid of set V. So it may be that there is a region of above average performance within cell **uy**, in the same way that the transition zone is within cell **ua**. We would have to create a zoomed-in gird within **uy** to find out; we have not done so.

7-5. Summary of Three-Process Work

7-5.1. Summary and Conclusions Concerning 2x2 and 3x3

With three processes, there are three arrangements and five configurations: APART, TOGETHER, and three loner configurations; the loner configurations have two processes on one processing node and one process alone on another processing node. Experiment set U is based on the full grid, and experiment set V is based on the zoomed-in grid, both with three processes per trace.

In almost every run, APART is fastest, and the three loner-type configurations are second-, third-, and fourth-fastest. In every run, TOGETHER is slowest. On average, APART is about 36% faster than the fastest loner; there is about 11% difference between the fastest and the slowest loner; and TOGETHER is about 51% slower than the slowest loner. Altogether, the slowest configuration is a mean of 98% slower than the fastest. We conclude that not only is there significant potential benefit in avoiding the worst configuration, but also in selecting the very best configuration.

In both 2x2 and 3x3 situations, APART is overwhelmingly the fastest configuration. Only when there are less than 4% CPU execute actions is APART not fastest, and even then APART is only about 4% slower than the fastest configuration. In both situations, moderately successful predictions of the fastest configuration can be made based on CPU service demand, but in both it is faster and more effective to always use APART.

The transition zone (the region in which APART is not always the best configuration) is slightly smaller in 3x3 set V than in 2x2 set R. If it continues to shrink as the number of processes grows, the always-APART policy may be even better for larger numbers of processes.

Generalizing, it appears that an APART configuration will be fastest in nearly any NxN situation, the exceptions being unrealistic cases in which the amount of communication is huge and the amount of computation small.

APART configurations use all available processing nodes, and spread the processes out as much as possible. Generalizing, we suppose that in NxM situations the fastest configurations will be those which make use of all M processing nodes, thus spreading out the processes as much as possible. For example, arrangements of the form 1:1:...:1:1 should be fastest when there are N processing nodes, and arrangements of the form 2:1:1:...:1:0 should be fastest when there are $N-1$ processing nodes.

The maximum number of processes collocated on any processor is a major influence on the response time of a configuration. Two 1-collocated configurations (2x2 APART and 3x3 APART) have similar response times and two groups of 2-collocated configurations (2x2 TOGETHER and the 3x3 loners) have similar response times. Based on the two-process and three-process experiments, the mean difference in response time between a 1-collocated and a 2-collocated configuration is about 30-50%, although individual differences can be from less than 1% to almost 100%. The mean difference is much less than double, indicating that there is considerable overlap in processes' use of the resources of a processing node. Based only on the three-process experiments, the mean difference between a 2-collocated and a 3-collocated configuration is about 51-62%, and the mean difference between a 1-collocated and a 3-collocated configuration is about 98%. Overall, and not terribly precisely, it appears that adding a processing node improves response time by very approximately 50%.

The collocation-number result also indicates that an APART configuration should be fastest in any NxN situation, but it does not indicate whether an arrangement of the form 2:1:1:...:1:0 or one the form 2:...:2:0:...:0 would be fastest when there are $N-1$ processing nodes. Indurkhya, Stone, and Xi-Cheng's [ISX86] and Nicol's [Nic89] results, described in Section 2-3.2, indicate that the 2:...:2:0:...:0 arrangement is likely to be faster. We will explore this issue further in Chapter 8.

7-5.2. Three Processes with Two Processors (3x2)

7-5.2.1. Introduction and Response Time

When there are three processes to execute but only two processing nodes available, a 3x2 situation, APART cannot be used and TOGETHER is always slowest, so the fastest 3x2 configuration will be one of the 2:1 loners. The question, then, is how to identify the fastest of the loner configurations. In set U, the mean difference between the response times of the loner configurations is 5-6%, and the differences get as large as 27%. It is worth a certain amount of effort and placement decision-making delay to try to obtain a mean 5-6% improvement in response time. The mean differences between loner configurations in set V are only about 2%, although the largest differences are almost 11%.

7-5.2.2. Matching in Set V

One way to try to predict the best loner process is to pick the process that has the least, or the most, of some measurable quantity. For example, the process with the least communication activity might be the best loner, or the process with the most CPU activity might be the best loner. The processes of a distributed program can be ordered by increasing or decreasing values of some variable, and that variable order compared to the loner order of which loner configuration is fastest, second-fastest, and slowest. When the orders are identical, we call it an all-match; when the predicted fastest loner matches the actual fastest loner, we call it a best-match. We count the number of best-matches and all-matches for each of several variables and directions (most-is-best or least-is-best), using data from APART and TOGETHER simulations, using two approaches to dealing with processes that have identical variable values (eql and noeql).

Matching on least-commtime successfully predicts the full loner order in about 48% of the runs in set V, which is quite good, and selects the best loner configuration in about 64% of the runs in set V. Random choice should select the best loner about 33% of the time, so matching is about twice as good as random choice in set V.

Being successful in 64% of the runs in set V is not the same as being successful 64% of the time. Real distributed programs are probably not uniformly distributed over the zoomed-in grid of set V, indeed they may not fall into the zoomed-in grid at all.

A placement procedure based on least-commtime matching would pick the middle-best loner in about 20% of the runs in set V, and the worst loner configuration in about 15% of the runs in set V. So even when matching misses choosing the best loner, it still performs better than would a random choice between the other two processes.

Matching performs slightly better outside the transition zone than in it, the success rate outside the transition zone rising to about 67% of the runs. On the other hand, none of the the slicing analyses showed a pattern of continuing improvement.

Matching with two variables gives no useful advantage over using only one variable.

7-5.2.3. Attractive and Repulsive Forces in Set V

Another approach to choosing the best loner configuration in a 3x2 situation is to look at the ‘‘forces’’ which ‘‘attract’’ processes toward each another and ‘‘repel’’ them away from one another. Two processes which communicate frequently or voluminously may benefit from being placed together on the same processor; we can say that they are ‘‘attracted’’ to each other, and that there is an attractive force between them, like magnetism, whose strength is proportional to the amount of communication between them. Similarly, if two processes both have heavy CPU demands, they may benefit from being placed on separate processors; we can say that they ‘‘repel’’ each other, and that there is a repelling force between them, again like magnetism, whose strength is proportional to the amount of CPU time consumed.

We have investigated three attractive forces, three different ways of measuring the attractive force of communication. We have investigated one measure of the repulsive force: the sum of the elapsed CPU time for each pair of processes. We have investigated one measure which combines attractive and repulsive forces: the combined transmission time and CPU time of each pair of processes.

For each run we make best-loner predictions according to each of these five approaches, and one using most-cputime matching. Each prediction is compared to the actual loner response times; the result for each prediction is ‘‘best,’’ ‘‘mid,’’ or ‘‘worst,’’ depending on whether the process selected as best loner is actually the best loner, middle-best loner, or worst loner, respectively.

The three attractive-force approaches each select the best, middle, and worst loner process about one-third of the time. In other words, these approaches do no better than randomly choosing a loner process. The fact that the three approaches have very similar results suggests that it really does not matter how we measure the attractive force.

The repulsive-force approach is more successful than the attractive-force approaches, but less so than the combined approach.

The combined attraction and repulsion approach has the best success rate: 62.5% bests, about 26% mids, and only about 12% worsts. This is certainly much better than random selection of a loner configuration, but it is not as high as we would like.

The best success rates of matching and attractive/repulsive forces are very close. The difference is probably not significant, so we cannot choose between the two techniques based on the success rates in set V. The matching technique would be somewhat less expensive to use, in that it requires only one type of data, whereas attractive/repulsive forces requires two.

7-5.2.4. Set U as 3x2

We carried out matching and attractive/repulsive forces analyses in full-grid set U.

The best result for the matching technique in set U is about 83% successes using the eql approach; this is achieved by selecting as the loner process the process with the largest CPU demand in the APART simulations. The best result for the attractive/repulsive forces technique in set U is about 77% successes; this can be achieved by using any of the approaches involving the repulsive force of CPU demand, using data from the TOGETHER simulations. We believe that the success rates of the repulsive and combined results using APART data would be the same 83% as matching on most-cputime using APART data. Based on this supposition, we can say that, in set U, the best matching results and the best attractive/repulsive results are probably about the same. The matching technique or the repulsive forces technique would be somewhat less expensive to use, in that either of them requires only one type of data, whereas the combined attractive and repulsive forces technique requires two.

It appears that, over the full grid, CPU time is the most important factor in selecting the best configuration. In the zoomed-in grid, communication increases in importance, but CPU time is still more important.

7-5.3. Regions of the Grid

7-5.3.1. The Sci and VSci Regions

We believe that most real distributed programs fall into the lower left part of the full grid. Real distributed programs are probably large and expensive to run, otherwise they would not be worth distributing; in other words, most real distributed programs are scientific programs. These programs probably have a very high proportion of CPU use, and do little communication or I/O. To the extent that they do either, they probably do more communication than I/O. Most likely a parent process or master process reads in whatever input information is needed, then distributes it to the child or slave worker processes by sending messages. During the computation, the worker processes would have to communicate among themselves, to synchronize, to share intermediate results, and so on. The region thus described is in the lower part of the full grid, where the proportion of CPU execute actions is high, and on the left side, where the ratio of communication to I/O is high.

We define a region of seven cells in the lower left of the full grid which we call the scientific region, or sci region. We also define a smaller region of three cells in the extreme lower left of the full grid which we call the very scientific region, or vsci region. In the sci and vsci regions, the success of the attractive/repulsive forces technique (71% and 83%, respectively) is not as good as the success of the matching technique (79% and 83-100%).

The success rates of both the matching and attractive/repulsive forces techniques are about the same or slightly higher in the sci and vsci regions than in all of set U. If real distributed programs do actually fall into the sci or vsci regions of the grid, we can expect that the techniques we have explored for dealing with 3x2 situations will have approximately the success rates we found in set U, or slightly better.

7-5.3.2. Experiment Set W

To further explore to the scientific, high-computation area, we zoomed in on cell **uz**, creating an intermediate-scale grid with increments of five percentage points vertically and horizontally. The experiment set based on this grid and with three processes per program is set W.

APART is the best configuration in every run in set W, so the always-APART policy is perfectly successful. Dyadic placement also selects APART for every run.

The best success rates for matching and attractive/repulsive forces in set W are about 84%. That is, it is possible to select the best 3x2 loner configuration in about 84% of runs for scientific (heavy CPU, more communication than I/O) distributed computations. These success rates are about the same as those for the sci and vsci regions and for set U as a whole. There does not seem to be a trend as we move deeper into the lower left-hand corner.

7-5.3.3. The LR Region

The success rates of various techniques for selecting the best configuration are ‘‘below average’’ in the zoomed-in grid in the upper left corner compared to the ‘‘average’’ success rates over the full grid. We investigate the possibility of an ‘‘above average’’ area symmetrically located in the lower right. We define a region of six cells in the lower right of the set U full grid which we call the LR region.

Neither the matching approach nor the attractive/repulsive forces approach have higher success rates in LR than they do over the full grid. It is clear that LR is not an above average region. However, we consider the upper left to be below average because of the transition zone, which appears inside the highly zoomed-in grid. Similarly, it may be that there is a region of above average performance within lower right cell **uy**, in the same way that the transition zone is within cell **ua**.