Grouping MPI Processes for Partial Checkpoint and Co-migration

Rajendra Singh and Peter Graham*

Dept. of Computer Science
University of Manitoba
Winnipeg, MB, Canada, R3T 2N2
{rajendra,pgraham}@cs.umanitoba.ca

Abstract. When trying to use shared resources for parallel computing, performance guarantees cannot be made. When the load on one node increases, a process running on that node will experience slow down. This can quickly affect overall application performance. Slow running processes can be checkpointed and migrated to more lightly loaded nodes to sustain application performance. To do this, however, it must be possible to; 1) identify affected processes and 2) checkpoint and migrate them independently of other processes which will continue to run.

A problem occurs when a slow running process communicates frequently with other processes. In such cases, migrating the single process is insufficient. The communicating processes will quickly block waiting to communicate with the migrating process preventing them from making progress. Also, if a process is migrated "far" from those it communicates with frequently, performance will be adversely affected.

To address this problem, we present an approach to identify and group processes which we expect to be frequent communicators in the near future. Then, when one or more process is performing poorly, the entire group is checkpointed and co-migrated. This helps to improve overall application performance in shared resource environments.

1 Introduction

Using idle compute resources is cost-effective and systems like Condor have successfully exploited such resources in limited contexts (e.g. parameters sweep studies). Increasingly, networks in large organizations are becoming more capable and, when combined with latency tolerance mechanisms, can now provide an attractive platform for running message passing parallel programs as long as the inter-process communication is not intensive.

In environments where machines are shared, however, load guarantees cannot be made. If one or more machines become overloaded it will decrease application performance. This provides a strong motivation to be able to checkpoint and migrate the affected processes to new machines. Such *performance-driven* migration should not involve the entire set of application processes as this would be wasteful both in terms of lost progress and overhead (from migrating processes).

* Supported by the Natural Sciences and Engineering Research Council of Canada.

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 69–80, 2009.

We previously extended LAM/MPI to provide *partial* checkpoint and migration [1]. Our system checkpoints only those MPI processes that *need* to migrate due to overloading on their host nodes. For such partial checkpoint and co-migration to be effective, however, inter-process communication patterns must be considered which is the focus of this paper. Our prototype instruments the LAM/MPI Request Progression Interface (RPI) to efficiently gather pair-wise inter-process communication events. This data forms the basis on which pattern discovery can be done to determine inter-process communication patterns. These patterns are then used to predict the communication patterns expected in the near future, following a checkpoint event. The predicted patterns, in turn, provide the information needed to determine groups of processes that are expected to communicate frequently with one another. If one process in such a group needs to be migrated then all the processes in that group are checkpointed and co-migrated to a new set of lightly-loaded nodes. Thus, we ensure that these processes will be located near each other and thus continue to run effectively.

The rest of this paper is organized as follows. In Section 2 we briefly overview our prototype system for partial checkpoint and migrate. We then describe our approach to discovering communication patterns in Section 3. Section 4 explains how we predict groups of future frequently communicating processes using the discovered patterns. Some results of using our prediction algorithms applied to synthetic communications data are given in Section 5. We then briefly review related work in Section 6. Finally, in Section 7, we present some conclusions and discuss our planned directions for future work.

2 Our Partial Checkpoint/Migrate System

We have developed a prototype system for *partial* checkpoint and migration of MPI applications in a LAM/MPI environment that builds on LAM/MPI's SSI (System Software Interface) plugin architecture and the Berkeley Lab Checkpoint and Restart (BLCR) facility [2]. Our system [1] enables checkpoint, migration and restart of only those processes that have been impacted by overloaded compute nodes. A fundamental challenge in doing this is to be able to allow unaffected processes to continue their execution while the affected processes are being checkpointed, migrated and restarted. We have modified the LAM/MPI code base slightly to allow this and also to efficiently gather inter-process communication information on a pair-wise basis.

In our earlier work, we sampled communication information periodically and maintained summary information over multiple time scales (e.g. 1, 10 and 100 time units). This information was then weighted in various ways to try to determine the frequently communicating process groups. In our initial implementation we naively weighted recent information over older information and defined a simple threshold for deciding whether or not two processes were communicating frequently enough that they should be checkpointed together. Not surprisingly, the effectiveness of this simple grouping strategy was limited. In this paper we present new approaches to predicting future communication patterns that allow

us to better group frequently communicating processes for checkpointing and co-migration and we assess the effectiveness of the groupings.

3 Discovering Communication Patterns

MPI applications exhibit varying communication patterns over time. For example, processes in an FFT computation will communicate with other processes as determined by the "butterfly" exchange pattern inherent in the algorithm. A process P_i will communicate with different processes depending on the phase of the computation. We need to be able to discover and group frequently communicating processes accordingly. Naturally, applications where all processes communicate frequently will be unable to benefit from our work.

We predict future communication patterns based on previously observed patterns. Using the predicted patterns, we identify groups of processes that communicate frequently so that, when necessary, they can be checkpointed and co-migrate together. The basis of any prediction scheme relies on our ability to detect communication patterns using the data on pair-wise communication events collected by our system. We considered three possible approaches to identifying recurring patterns in the data: machine learning (e.g. the construction and use of Hidden Markov Models [3]), data mining techniques (e.g. sequential mining [4,5]) and various partial match algorithms.

Knowing that communication patterns between parallel processes sometimes vary and wanting to be able to reason about the pattern matching that is done so we could "tune" our approach, we decided to explore partial match algorithms. The TEIRESIAS [6] pattern matching algorithm is used for a number of bioinformatics problems related to sequence similarity – how "close" one sequence is to another. Of interest to us, TEIRESIAS is able to find long recurring patterns in sequences and is also able to do this subject to slight differences in the patterns. Identifying these types of patterns is important to determining groups of communicating processes. Relatively long, repeating patterns of nearly identical pair-wise communication events form the basis for determining which groups of processes can be reliably expected to communicate with each other.

When applied to DNA nucleotide sequences, the input alphabet, Σ, used by TEIRESIAS would simply be: { A,C,T,G } and the recurring patterns discovered would be sequences of these symbols interspersed with some, relatively small, number of "don't care" symbols representing minor variations in the repeating patterns. To use TEIRESIAS to discover patterns in inter-process communication we must define an alphabet based on the communications data. Each pair of processes is mapped to a unique symbol using the mapping function: $\Theta < p1, p2 >= (p1 \times NumProcs) + p2$, where $p1$ and $p2$ are the process ranks, and $NumProcs$ is the number of MPI processes. TEIRESIAS can support up to $2^{31} - 1$ symbols and its running time is independent of the number of symbols.

The output from our implementation of TEIRESIAS is a list ordered by length of commonly occurring patterns in an input sequence of communication events. Each such pattern is accompanied by a list of offsets identifying where they

occur in the input sequence. This offset information is used to identify patterns immediately preceding where a checkpoint occurs. It is also useful for identifying points at which communication patterns change.

4 Predicting Communicating Groups of Processes

Given the frequently occurring patterns discovered by TEIRESIAS in the inter-process communication data we collect in our modified version of LAM/MPI, our goal is to be able to respond to a need to checkpoint and migrate by accurately predicting which processes should be grouped together based on expected inter-communication. This can be done in a number of ways, subject to certain characteristics of our proposed environment.

4.1 The Prediction Environment

TEIRESIAS, while efficient, like other pattern discovery techniques, can be computationally expensive for complex patterns and long sequences. The communication data collected by our instrumented LAM/MPI implementation is periodically copied asynchronously to the TEIRESIAS-based pattern discovery process which runs in parallel with the `MPIRUN` process that starts and coordinates the processes forming each running MPI application. Ideally, and not unrealistically, these two "control" processes will run on a very capable host system.

The collection, transfer and analysis of communication patterns occurs on a regular basis, defining fixed length "sampling periods". The need to checkpoint processes due to low performance occurs at unpredictable times and therefore asynchronously with respect to the sampling periods. Predictions about communicating process groups in the current sampling period must be made based on patterns extracted from preceding sampling periods.

4.2 Inter-Process Communication Characteristics

Message passing parallel applications can exhibit very different communications behaviour and resulting inter-process communication patterns. Application communication behaviour may depend on the parallel programming style used, the characteristics of the data being processed or on specifics of the selected algorithm(s). In many cases, however, there is a good deal of regularity in communications that can be exploited to identify groups of communicating processes.

For example, in Master/Slave parallel programming, the pattern of communication between the master and slaves, and among slaves is normally recurring over time. A typical Master/Slave approach involves three phases: data distribution, computation and data collection. The master distributes data to the slaves/workers in equal chunks. The workers then work on the data and, if required, communicate with other worker processes to exchange information. After computing, the workers send results to the master. This entire process repeats.

The use of Finite Element Methods (FEMs) for heat transfer problems is a good example of a calculation leading to recurring communication based on the structure of the data being processed. In such problems, modeling of the physical state of the entire system is partitioned and solved in parts across parallel processors. To calculate the temperature in one partition, the computing process needs to know the temperature in its neighboring partitions. The structure of the data partitioning thus determines the pattern of inter-process communication. In this kind of computation, the pattern of communication recurs for each time step that the application is executing.

Communication patterns may also be algorithm induced. An example of such an application is the computation of the Fast Fourier Transform (FFT) using the well known butterfly approach. This example results in changing but regular patterns of inter-process communication.

4.3 Possible Types of Predictors

The space of possible predictors can be divided into three major types as shown at the top of Fig. 1. A general, application-independent predictor will be simple but may be unlikely to give consistently good results for a range of applications. Custom, application specific predictors offer the best potential for accuracy but will require significant effort from application programmers who will likely have to create the predictors themselves. Instead, the use of a small number of generally useful predictors for a range of application types may be possible. These "specific" predictors will, in some sense, be application dependent but in a very general way, based on an application's inter-process communication characteristics (such as those described previously). Using such an approach, application developers will be able to easily pick from a small set of available predictors based on the high-level characteristics of their applications. In this paper, we explore some possible predictors for what are perceived as some typical application types and then assess their performance.

4.4 Our Initial Predictors

We began by implementing two experimental predictors: a *basic* predictor that attempted to predict process groups based only on very recent communication

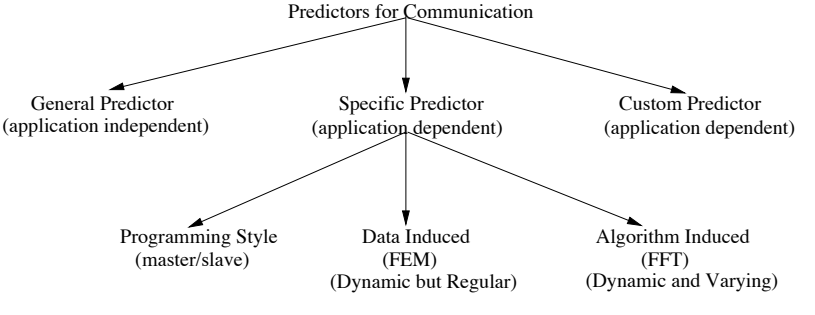

Fig. 1. Predictor Space

patterns, and a *historical* predictor which exploits information about communication patterns from earlier sampling periods, if available.

Our basic predictor selects the longest possible frequently occurring pattern in the most recent sampling period that has occurred at-least twice before the checkpoint arrival time and which is closest (in terms of time) to the time of checkpoint request. This is determined using the *offset* list and length of each pattern discovered by TEIRESIAS in that sampling period. If we are able to find a pattern meeting the criteria we designate it to be the basic prediction. This makes intuitive sense since it is the pattern of communication that appears to be on-going and is therefore, hopefully, likely to continue. If we don't find any pattern that meets these criteria then we simply pick the last pattern found from the previous sampling period and designate it as the basic prediction. This basic predictor serves as a baseline for the other predictors and is also used whenever the historical predictor cannot be used.

Our historical predictor was intended to address possible shortcomings of the basic predictor based on the fact that it only uses recent communication patterns to make a prediction. In the case where the application communication pattern is changing at the time of checkpoint, the immediately preceding patterns may not be the basis of a sound prediction. Our historical predictors identify patterns in the recent behaviour of the application and attempt to match these patterns against, possibly, older patterns stored from previous sampling periods. In this way, if the application is starting a new pattern of communication that has been seen earlier, we can make a more appropriate prediction based on the patterns from the earlier sampling period(s). This can be very effective for applications, like FFT, with varying but recurring communication patterns.

We implemented two versions of the historical predictor, one which makes a historical prediction only if there is an **exact** prefix match with a pattern from a previous period and the other which may accept a "close" historical match based on Levenshtein distance [7]. These are referred to as the *specific* and *Levenshtein* historical predictors, respectively.

Using the specific historical predictor, the pattern from the basic predictor is used as input and checked for an exact match against stored patterns seen in the past. If we find an exact match against patterns from one or more preceding sampling periods, then the pattern that occurred in the past immediately after the matched historical pattern becomes the predicted pattern. In the special case where the basic prediction exactly matches a pattern that occurred previously at the end of a sampling period, then we pick the first pattern that occurred in the next historical sampling period to be the predicted pattern. This process, naturally, chooses the communication pattern we expect to see next as the predicted pattern. If there is no match found then the specific historical predictor fails and the basic prediction is used.

The Levenshtein historical predictor is based on the concept of Levenshtein distance which simply reflects the number of mismatches between two compared patterns. The Levenshtein predictor works much as the specific predictor does except that it can make predictions when no exact match is found. We start with

the basic prediction and go through the stored historical patterns and compute the Levenshtein distance between the base prediction and each pattern. We select the pattern with the minimum Levenshtein distance (beneath an acceptability threshold) and designate its historical successor to be the predicted pattern.

4.5 Our Enhanced Predictors

We conducted a preliminary series of experiments (described in Section 5) using our initial prediction strategies. While the predictors generally performed reasonably well across a range of synthetically generated communication data, some aberrant behaviour was noted. We determined that the unexpected behaviour arose because TEIRESIAS was matching long patterns consisting of multiple repetitions of the same "fundamental" pattern. For example, given a frequently repeated inter-process communication pattern, A, TEIRESIAS was generating patterns such as AA, AAA, $AAAA$, etc. This was because the fundamental pattern (A in this case) occurred multiple times in immediate succession within a single sampling period. By favouring longer patterns, the fundamental pattern was ignored. Instead, a pattern close in size to the sampling period consisting of many repetitions of the fundamental pattern was selected. Unfortunately, this excessively long pattern did not reoccur frequently until the application had been running for some time. This resulted in poor performance when checkpoints were requested relatively early in an application's execution.

To solve this problem, an enhanced prediction algorithm was created that explicitly checks for such repeated sequences, allowing it to identify the underlying fundamental patterns that are best used for prediction purposes. We use the Knuth-Morris-Pratt (KMP) pattern matching algorithm [8] to identify fundamental patterns by finding adjacent repeated patterns in the long patterns discovered by TEIRESIAS which consist solely of repetitions of a shorter pattern discovered by TEIRESIAS. The fundamental patterns identified in this way are then used in place of the original set of patterns found by TEIRESIAS.

5 Results

To assess the effectiveness of our predictors in a controlled environment, we chose to experiment using synthetically generated communications data. To ensure meaningful results, our synthetic data was created to reflect well-known communication patterns. This allowed us to start with clean and consistent patterns where we could more easily understand the behaviour of our predictors. We use the Levenshtein distance between our predicted pattern and the sequence seen in the synthetic data as our measure of goodness. We experimented with data reflecting the key communications characteristics of the three different sample application types discussed earlier: master/slave, FEM-like and FFT-like. Knowing that patterns in real communications data will vary, we introduced noise into and between the regular communications patterns for the FEM-like data to ensure that TEIRESIAS would still discover the necessary patterns. We created

hand-crafted data for a variety of "application sizes" and found consistent results independent of the number of processes. We also found that TEIRESIAS does tolerate slight differences in and between recurring patterns. We describe a subset of our experiments highlighting key results. In the figures that follow, 'basic' refers to predictions made using only recent communications behaviour, 'historic' refers to predictions using the exact match historic predictor and 'levenshtein' refers to predictions made using the approximate match (Levenshtein distance) historic predictor. Results for our enhanced predictors which use the KMP algorithm to identify fundamental patterns have a '(kmp)' suffix. Result graphs are presented in 3D only where it improves feature identification.

Figure 2 shows the results using our initial prediction schemes (i.e. no identification of fundamental patterns) for an application with master/slave style communication pattern. The basic predictor, not unexpectedly, performs poorly, failing to correctly handle changes in communications patterns while the Levenshtein predictor performs surprisingly well. Of particular interest, however, is the drastic misprediction made by the historic predictor at, approximately, time 900 in this example run. This was caused because TEIRESIAS discovered a long pattern consisting of consecutive "fundamental" patterns which did not occur in the sampling period around time 900. This was the aberrant behaviour, described earlier, that lead us to develop our enhanced prediction schemes. Similar behaviour was also seen for other communication patterns (e.g. FFT-like).

As expected, our enhanced predictors perform significantly better than our initial ones, consistently providing high-quality prediction results for both master/slave and FFT-like communication patterns as can be seen in Fig. 3 and Fig. 4, respectively. Although graphs are only shown for the 8 process case, these results generalize to other problem sizes using more or less processes. They also are consistent with results for FEM-like communications with "noise". The unexpectedly good performance of our original Levenshtein predictor can be explained by the closeness of the patterns seen in each sampling period to those

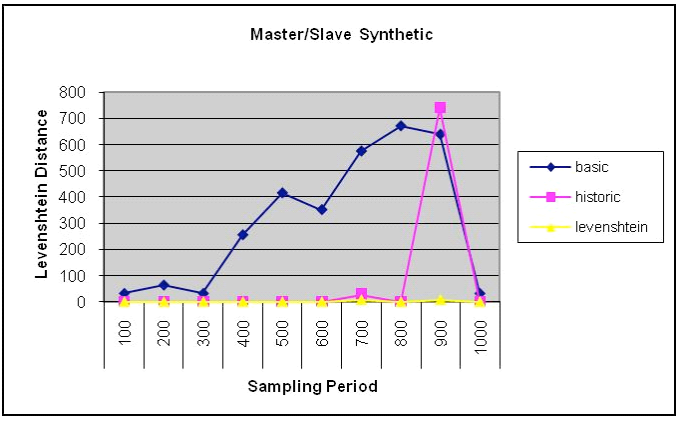

Fig. 2. Master/Slave Results for Original Predictors

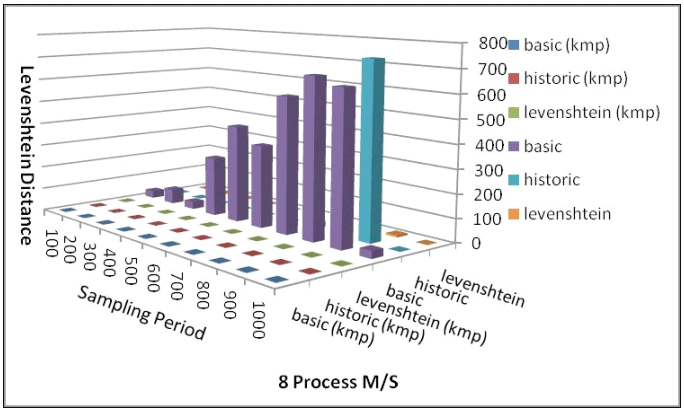

Fig. 3. Master/Slave Results for 8 Processes

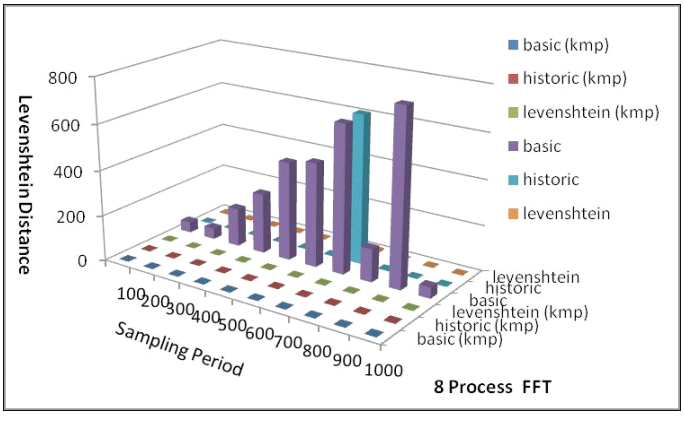

Fig. 4. FFT Results for 8 Processes

discovered by TEIRESIAS. We do not expect this performance to generalize well to real data and more complex communications patterns.

We also ran experiments to test the sensitivity of our predictors to the sampling period size. The results for the start of a four process master/slave communication pattern are shown in Fig. 5[1]. Not surprisingly, our original predictors perform very poorly. It is also noteworthy that our enhanced predictors also show some variation. Choosing an appropriate sampling period size will likely be important to successful prediction for some patterns of communication.

Finally, we did some experiments where the large-scale characteristics of the various communication patterns were varied. For example, we increased the

[1] We use ribbon graphs for clarity. This is not intended to suggest continuous data.

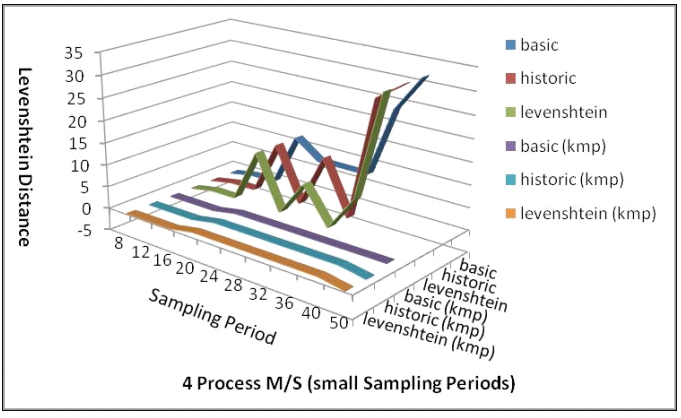

Fig. 5. Master/Slave Results with small sampling period for 4 Processes

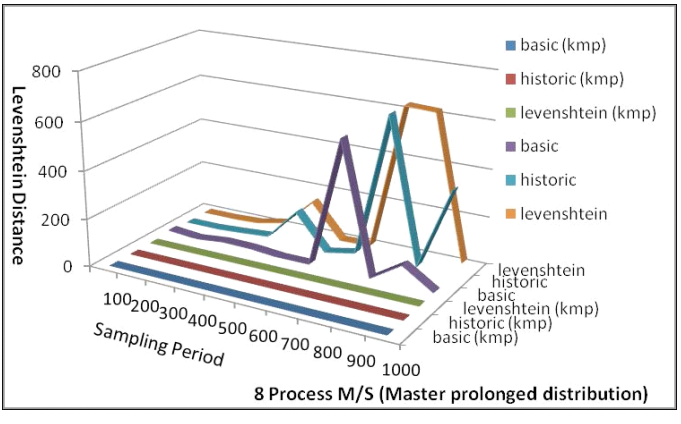

Fig. 6. Master/Slave Results with prolonged distribution phase for 8 Processes

duration of the data distribution phase for master/slave communications (refer to Fig. 6). In general, we found our enhanced predictors to be much more tolerant of a range of changes. This enforces our belief that there is value in determining fundamental patterns and using them for prediction.

6 Related Work

A number of systems have been built to add checkpointing to PVM, another message passing parallel programming system. These include Migratable PVM (MPVM) [9], and DynamicPVM [10] both of which support only full checkpointing. Vadhiyar et al [11] presented a migration framework designed for grid

environments and Huang, et al [12] described Adaptive MPI (AMPI) which provides "process" migration on top of and tightly coupled with Charm++. Most recently Wang, et al [13] have also developed a LAM/MPI partial checkpoint system. Unlike our system, theirs is focused on reliability rather than performance and does not permit any MPI processes to continue running during checkpoint and migration. In a large dedicated cluster environment (the focus of their work) this is not an issue since migrate and restart is fast and inter-process communication is uniform and efficient. In a shared environment such as the ones motivating our work, however, this is inefficient.

7 Conclusions and Future Work

We have developed a family of predictors that can be used to group frequently communicating MPI processes for checkpointing and co-migration to improve their performance on shared processors. Our predictors are based on (possibly refined) inter-process communication patterns discovered by the TEIRESIAS pattern discovery algorithm run against a sequence of pair-wise communication events captured by our LAM/MPI code enhancements. Our initial results using synthetic data that mimics known communication patterns are promising. Somewhat surprisingly, we saw relatively little difference between the various algorithms for different application types. This suggests that it may be possible to create a reasonably good "default" predictor that could be used where little is known about an application's communications characteristics or where novice programmers cannot pick a specific predictor. Even if the consistent performance is partially an artifact of the synthetic data, we now have confidence that a range of message passing applications can be supported by only a few predictors.

We plan to explore several directions for future work. So far, we have only tested our predictors using synthetic data but we are now collecting actual inter-process communication data and will use this to verify the results obtained and to increase our confidence in our ability to handle small changes in communications behaviour that occur in real programs. With a sufficiently large set of test applications we should also be able to identify optimizations that will better suit particular application communications characteristics. We will explore this in an attempt to ensure consistent prediction accuracy. While TEIRESIAS outperforms other partial match algorithms using its "convolution" approach [6], its worst-case running time is still exponential for complex patterns with many variations. Therefore, it is not practical to run TEIRESIAS too frequently. Further, our predictor performance is sensitive to the size of the sampling periods. For the synthetic data used and for dozens of processes, our predictor runtimes were all sub-second but as the number of communication events increases so too will the runtime. We are interested in how to balance computational overhead and prediction accuracy and will do experiments to determine how few pattern discoveries we can do before missing important changes in inter-process communications for different types of applications. We will then attempt to create a mechanism to adapt the sampling period dynamically. We are also interested in

finding ways to predict when communication pattern changes are about to occur to avoid occasional mispredictions. Finally, while TEIRESIAS has performed well on the synthetic data, we are prepared to explore other algorithms to deal with any unanticipated problems as we move to real data.

References

1. Singh, R., Graham, P.: Performance Driven Partial Checkpoint/Migrate for LAM-MPI. In: 22nd International Symposium on High Performance Computing Systems and Applications (HPCS), June 2008, pp. 110–116 (2008)
2. Duall, J., Hargrove, P., Roman, E.: The design and implementation of berkley lab's linux checkpoint/restart. Technical report, Berkley Labs LBNL-54941 (2002)
3. Brejova, B., Vinar, T., Li, M.: Pattern discovery: Methods and software. In: Krawetz, S.A., Womble, D.D. (eds.) Introduction to Bioinformatics, ch. 29, pp. 491–522. Humana Press (2003)
4. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalization and Performance Improvement. In: Int'l Conf. Extending Database Technology, pp. 3–17 (1996)
5. Pei, J., Han, J.e.a.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In: Int'l Conf. Data Engineering, pp. 215– 226 (2001)
6. Rigoutsos, I., Floratos, A.: Combinatorial pattern discovery in biological sequences: The teiresias algorithm. Bioinformatics 14(1), 55–67 (1998)
7. Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics 10, 707–710 (1966)
8. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast Pattern Matching in Strings. SIAM Journal on Computing 6(2), 323–350 (1977)
9. Casas, J., Clark, D., Konuru, R., Otto, S., Prouty, R., Walpole, J.: MPVM: A Migration Transparent Version of PVM. Technical report, CSE-95-002, 1 (1995)
10. Dikken, L., Linden, F.v.d., Vesseur, J., Sloot, P.: Dynamic PVM – Dynamic Load Balancing on Parallel Systems. In: Proceedings Volume II: Networking and Tools, pp. 273–277. Springer, Munich (1994)
11. Vadhiyar, S.S., Dongarra, J.J.: A Performance Oriented Migration Framework for the Grid. In: 3^{rd} International Symposium on Cluster Computing and the Grid, Tokyo, Japan, May 12-15, 2003, pp. 130–137 (2003)
12. Huang, C., Lawlor, O., Kalé, L.V.: Adaptive MPI. In: Rauchwerger, L. (ed.) LCPC 2003. LNCS, vol. 2958, pp. 306–322. Springer, Heidelberg (2004)
13. Wang, C., Mueller, F., Engelmann, C., Scott, S.: A Job Pause Service under LAM/MPI+BLCR for Transparent Fault Tolerance. In: Proc. of the Parallel and Distributed Processing Symposium, pp. 1–10 (2007)