Online Performance Observation of Large-Scale Parallel Applications

Allen D. Malony and Sameer Shende and Robert Bell
{malony,sameer,bertie}@cs.uoregon.edu
Department of Computer and Information Science
University of Oregon
Eugene, Oregon 97405, USA

Parallel performance tools offer insights into the execution behavior of an application and are a valuable component in the cycle of application development, deployment, and optimization. However, most tools do not work well with large-scale parallel applications where the performance data generated comes from upwards of thousands of processes. As parallel computer systems increase in size, the scaling of performance observation infrastructure becomes an important concern. In this paper, we discuss the problem of scaling and perfomance observation, and the ramifications of adding online support. A general online performance system architecture is presented. Recent work on the TAU performance system to enable large-scale performance observation and analysis is discussed. The paper concludes with plans for future work.

1. Introduction

The scaling of parallel computer systems and applications presents new challenges to the techniques and tools for performance observation. We use the term *performance observation* to mean the methods to obtain and analyze performance information for purposes of better understanding performance effects and problems of parallel execution. With increasing scale, there is a concern that standard observation approaches for instrumentation, measurement, data analysis, and visualization will encounter design or implementation limits that reduce their effective use. What drives this concern, in part, is the problem of measurement intrusion, and the fact that simple application of current approaches may result in more perturbed performance data. It is also clear that scaling of standard methods will raise issues of performance data size, the amount of processing time required to analyze the data, and the usability of performance presentation techniques.

Concurrently, there is an interest in the online observation of parallel systems and applications for purposes of dynamic assessment and control. With respect to performance observation, we think of *performance monitoring* as constituting online measurement and performance data access, and *performance interaction* as additional infrastructure for affecting performance behavior externally. Certainly, there are several motivations for online performance observation, including the control of intrusion via dynamic instrumentation or dynamic measurement. However, the influence of scaling must again be considered when evaluating the benefits of alternative approaches.

In this paper, we consider these issues with respect to profiling and tracing methods for online performance observation. Our main interest in this work is to understand how best to scale a parallel performance measurement model and its implementation, and to extend its functionality to offer runtime control and interaction. We present results from the development of scalable online profiling in the TAU performance system. We also have develop online tracing capabilities in TAU, but this work is discussed elsewhere.

2. Scaling, Intrusion, and Online Observation

As the starting point for understanding the influences of scaling on performance observation, it is reasonable to consider the standard methods for performance measurement and analysis: *profiling* and *tracing*. Profiling makes measurements of significant events during program execution and calculates summary statistics for performance metrics of interest. These profile analysis operations occurs at runtime. In contrast, tracing captures information about the significant events and stores that information in a time-stamped trace buffer. The information can include performance data such as hardware counts, but analysis of the performance data does not occur until after the trace buffer is generated. For both profiling and tracing, it is usually the case that the performance measurements (profile or trace) are generated and kept at the level of application threads or processes.

What happens, then, as the application scales? We consider scaling mainly in terms of number of threads of execution. In general, one would expect that the greater the degree of parallelism, the more performance data overall that will be produced. This is because performance is typically observed relative to each specific thread of execution. Thus, in the case of profiling a new profile will be produced for each thread or process. Similarly, tracing will, in general, produce a separate event sequence (and trace buffer) for each thread or process. Certainly, these consequences of scaling have direct impact on the management of performance data (profile or trace data) during a large-scale parallel execution. Scaling, it is expected, will also cause changes in the number, the distribution, and perhaps the types of significant events that occur during a program's run, for instance, with respect to communication. Furthermore, larger amounts of performance data will result in greater analysis time and complexity, and more difficulty in presenting performance in meaningful displays.

However, the real practical question is whether our present performance observation methods and tools are capable of dealing with these issues of scale. Most importantly, this is a concern for measurement intrusion and performance perturbation. Any performance measurement intrudes on execution performance and, more seriously, can perturb "actual" performance behavior. While low intrusion is preferred, it is generally accepted that some intrusion is a consequence of standard performance observation practice. Unfortunately, perturbation problems can arise both with only minor intrusion and small degrees of parallelism.

How scaling affects intrusion and perturbation is an interesting question. Traditional measurement techniques tend to be localized. For instance, thread profiles are normally kept as part of the thread (process) state. This suggests that scaling would not compound globally what intrusion is occuring locally, even with larger numbers of threads (processes). On the other hand, it is reasonable to expect that the measurement of parallel interactions will be affected by intrusion, possibly resulting in a misrepresentation of performance due to performance perturbation. The bottom line is that performance measurement techniques must be used in an intelligent manner so that intrusion effects are controlled as best as possible. But this must involve a necessary and well-understood tradeoff of the need for performance data for solving performance problems against the "cost" (intrusion and possible perturbation) of obtaining that data.

Online support for performance observation adds interactivity to the performance analysis process. Several arguments justify the use of online methods. Post-mortem analysis may be "too late," such as when the status of long running jobs needs to be determined to decide on early termination. There may also be opportunities for steering a computation to better results or better performance interactively by observing execution and performance behavior online. Some have motivated online methods as a way to implement *dynamic* performance observation where both instrumentation and measurement can be controlled at runtime. In this respect, online approaches may offer a means to better manage performance data volume and measurement intrusion. Most of the arguments above assume, of course, that the online support can be implemented efficiently and results in little intrusion or perturbation of the parallel computation. This is more difficult with online methods as they involve more directly coupled mechanisms for access and interaction. Again, one needs to understand the tradeoffs involved to make an intelligent choice of what online methods to use and how.

3. Online Performance Observation Architecture

The general architecture we envision for online performance observation is shown in Figure 1. The online nature is determined by the ability to *access* the performance data during execution and make it available to analysis and visualization tools, which are typically external. Additionally, performance interacton is made possible through a performnace control path back into the parallel system and software. Here, instrumentation and measurement mechanisms may be changed at runtime.

How performance data is accessed is an important factor for online operation. Different access mod-

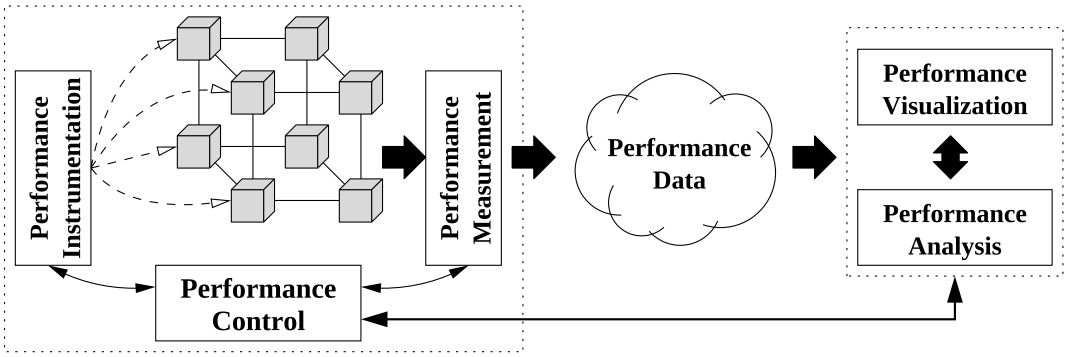

Figure 1. Online Performance Observation Architecture

els are possible with respect to the general architecture. A *Push* model acts as a producer/consumer style of access and data transfer. The application decides when, what, and how much data to send. It can do so in several ways, such as through files or direct communication. The external analysis tools are consumers of the performance data, and its availability can be signalled passively or actively. In contrast, a *Pull* model acts as a client/server style of access and transfer. Here, the application is a performance data server, and the external analysis tool decides when to make requests. Of course, doing so requires a two-way communication mechanism directly with the application or some form of performance control component. Combined *Push/Pull* models are also possible.

Online profiling requires performance profile data, distributed across the parallel application in thread (process) memory, to be gathered and delivered to the profile analysis tool. Profiling typically involves stateful runtime analysis that may or may not be consistent at the time the access is requested. To obtain valid profile data, it may be necessary to update execution state (e.g., callstack information) or make certain assumptions about operation completion (e.g., to obtain communication statistics). Assuming this is possible, online profiling will then produce a sequence of *profile samples* allowing interval-based and multi-sample performance analysis. The delay for profile collection will set a lower bound on interval frequency. This delay is expected to increase with greater parallelism.

Similarly, online tracing requires the gathering and merging of trace buffers distributed across the parallel application. The buffers may be flushed afterwards, thereby allowing only the last trace records since the last flush to be read. Such interval tracing may require "ghost events" to be generated before first event and after the last event to make the trace data consistent. If the tracing system dynamically registers event identifiers per execution thread, it will be necessary to make these identifiers uniform before analysis. (Static schemes do not have this problem, but require instead that all possible events be defined beforehand.)

4. Online Profiling in TAU

We have extended the TAU performance system [12] to support both online profiling and tracing. Given space constraints, we only describe our approach to online profiling is this paper. Our online tracing work can be found in [13].

4.1. Approach

The high-level approach we have taken for online parallel profiling is shown in Figure 2. The TAU performance system maintains profiling statistics on a *context* basis for each thread in a context [12]. Normally, TAU collects performance profiles at the end of the program run into profile files, one for each thread of execution. For online profiling, TAU provides a "profile dump" routine that, when called by the application, will update the profile statistics for each thread, to bring them to internally consistent states, and then output the profile data to files.

The performance data access model we have implemented and used in TAU is a Push model. The application scenario we want to target is one where there are major phases and/or interations in the

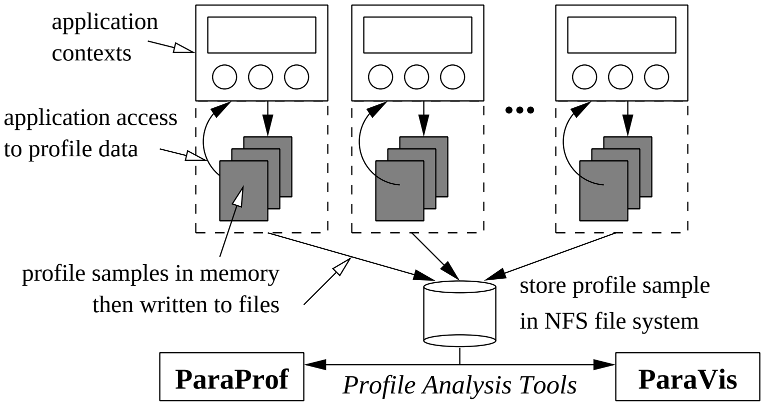

Figure 2. Online Profiling in TAU

computation where one would like to capture the current profile at those time steps. Thus, at these points, the application calls the TAU profile dump routine to output the performance state. Each call of the dump routine will generate a new set of profile files or append to files containing earlier profile dumps. The updating of the profile dump files is used to "signal" the external profile analysis tools.

One of the advantages of this approach is that it can be made portable and robust. The only requirement is support for a shared file system, using NFS or some other protocol. It is possible to implement a push model in the TAU performance system using a signal handler approach, but it introduces other system dependencies that are less robust.

A valid argument against this approach is that it has problems when the application scales, as the number of files increase and the file system becomes the bottleneck. There are four mechanisms we are investigating to address this problem. First, thread profiles for a context can be merged into a single context profile file. This directly reduces the number of files when there are multiple threads per context. Second, the profile dump routine allows event selection, thereby reducing the amount of profile data saved. The third mechanism is to utilize a data reduction network facility, such as Wisconsin' MRNet [6], to gather and merge thread/context profiles using the parallel communication hardware, before producing output files. This can both address problems with scaling file systems and problems with large number of files, by merging profile data streams in parallel until generating profile output files. Finally, the fourth mechanism is to leverage the more powerful I/O hardware and software infrastructure in the parallel system that one would expect to be present in the system as it is scaled (e.g., parallel file system, multiple I/O processors, clustered file system software, etc.).

4.2. Tools

Our work has produced two profile analysis tools that can be used online. ParaProf [8] is the main TAU tool used for offline performance analysis. It is capable of handling profiles from multiple performance experiments and gives various interactive capabilities for data exploration. ParaProf can accept profile data from raw files, our performance database, or through a socket-based interface.

The ParaVis tool [9] was developed to experiment with scalable performance analysis and visualization using three-dimensional graphics. The ParaVis arcitecture is shown in Figure 3; see [9] for details. A key result from our work with this tool is the importance of selection and focus in different components of the tool. Also, use of the tool demonstrated the online benefits of being able to see how performance behavior unfolds during a computation.

4.3. Application Access to Profile Data

Part of the online performance observation model includes the possibility of the application itself accessing its measured performance data. This is presently supported in TAU at the context level, where a thread can request the profile data for some measured event. Our intention is to extend this capability to include access to performance data on remote application contexts.

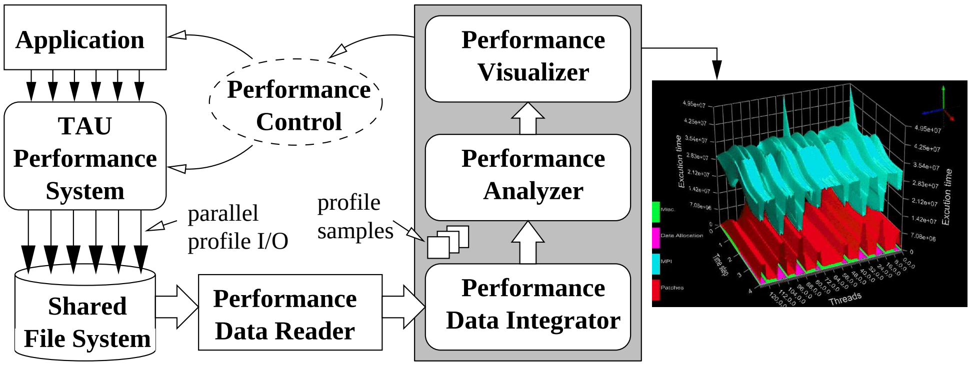

Figure 3. ParaVis Online Profile Analysis and Visualization

5. Related Work

The research ideas and work presented here relate to several areas. There has been a long time interest in the monitoring of parallel systems and aplications. This is due to the general hypothesis that by observing the runtime behavior or performance of the system or application, it is possible to identify aspects of parallel execution that may allow for improvement. Several projects have developed techniques that allow parallel applications to be responsive to program behavior, available resources, or performance factors. The Falcon project [3] is an example of *computational steering* systems [15] that can observe the behavior of an application and provide hooks to alter application semantics. These "actuators" will lead to changes in the ongoing execution. Because computational steering systems enable direct interaction with the application, they are often developed with visualization frontends that provide graphical renderings of application state and objects for execution control.

Online performance observation systems look to achieve several advantages for performance analysis. Paradyn [7] works online to search for peformance bottlenecks, while controlling the measurment overhead by dynamically instrumenting only those events that are useful for testing the current bottleneck hypothesis. Thus, the performance analysis done by Paradyn at runtime both collects profile statistics and interprets the performance data to decide on the next coarse of action. Where as Paradyn attempts to identify performance problems, Autopilot [1] is an online performance observation and adaptive control framework that uses application sensors to extract quantitative and qualitative performance for automated decision control. While both Paradyn and Autopilot are oriented towards automated performance analysis and tuning, neither address the problem of scalable performance observation or provide capabilties to analyze or visualize large-volume performance information.

Indeed, the difficulty of linking application embedded monitoring to data consumers will ultimately determine what amount of runtime information can be utilized. This involves a complicated tradeoff of instrumentation and measurement granularity versus the overhead of application / performance data transport versus the information requirements for desired analysis [4]. Projects such as the Multicast Reduction Network (MRNet) [6] will help in providing efficient infrastructure for data communication and filtering. Similarly, the Peridot [10] project is attempting to develop a distributed application monitoring framework for shared-memory multiprocessor (SMP) clusters that can provide scalable trace data collection and online analysis. The system will have selective instrumentation and analysis control, helping to address node- and system-level monitoring requirements. A different approach to scalable observation is taken in [5]. Here, statistical sampling techniques are used to gain representative views of system performance characteristics and behavior.

In general, we believe the benefits seen in the application of online computation visualization and steering, itself requiring demanding monitoring support, could also be realized in the parallel performance domain. Our goal is to consider the problem of online, scalable performance observation as a whole, understanding the tradeoffs involved and designing a framework architecture to address them.

6. Conclusion

The combination of scalable performance observation and online operation sets a high standard for effective use of present day performance tools. Many performance systems are not built for scale and work primarily offline. Our experience is one of extending the existing TAU performance system to address problems of scale through improved measurement selectivity, new statistical clustering functions, parallel analysis, and three-dimensional visualization. In addition, online support in TAU is now possible for both profiling and tracing using a Push model of data access. We have demonstrated these capabilities for applications over 500 processes. However, it is by no means correct to consider our TAU experience as evidence for a general purpose solution. As with TAU, it is reasonable to expect that other traditional offline tools could be brought online under the right system conditions. In the wrong circumstances, the approaches may be ineffective either because they process larger volumes of data or require more analysis power. Any solution to scalable, online performance observation will necessarily be application and system dependent, and will require an integrated analysis of engineering tradeoffs that include concerns for intrusion and quality of performance data. Our goal is to continue to advance the TAU performance system for scalability and online support to better understand where and how these tradeoffs arise and apply.

REFERENCES

[1] R. Ribler, H. Simitci, and D. Reed, "The Autopilot Performance-Directed Adaptive Control System," *Future Generation Computer Systems,* **18**(1):175-187, 2001.

[2] H. Brunst, W. Nagel, and A. Malony, "A Distributed Performance Analysis Architecture for Clusters," International Conference on Cluster Computing, December 2003, to be published.

[3] W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, J. Vetter, and N. Mallavarupu, "Falcon: On-line Monitoring and Steering of Large-Scale Parallel Programs," *Proceedings of the 5th Symposium of the Frontiers of Massively Parallel Computing*, pp.422–429, 1995.

[4] A. Malony, "Tools for Parallel Computing: A Performance Evaluation Perspective," in *Handbook on Parallel and Distributed Processing*, J. Blazewicz, K. Ecker, B. Plateau, an D. Trystram (Eds.), 2000, Springer-Verlag, pp. 342–363.

[5] C. Mendes and D. Reed, "Monitoring Large Systems via Statistical Sampling," *LACSI Symposium*, 2002.

[6] P. Roth, D. Arnold, and B. Miller, "MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools," Technical report, University of Wisconsin, Madison, 2003.

[7] B. Miller, et al., "The Paradyn parallel performance measurement tool", *IEEE Computer* **28**(11), pp. 37–46, November 1995.

[8] R. Bell, A. Malony, and S. Shende, "ParaProf: A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis," International Euro-Par Conference, pp. 17–26, August 2003.

[9] K. Li, A. Malony, R. Bell, and S. Shende, "A Framework for Online Performance Analysis and Visualization of large-Scale Parallel Applications," International Conference on Parallel Processing and Applied Mathematics, Czestochowa, Poland, September 2003, to be published.

[10] K. Fuerlinger and M. Gerndt, "Distributed Application Monitoring for Clustered SMP Architectures," accepted to *EuroPar 2003*, Workshop on Performance Evaluation and Prediction, 2003.

[11] D. Reed, C. Elford, T. Madhyastha, E. Smirni, and S. Lamm, "The next frontier: Interactive and closed loop performance steering," *Proceedings of the 25th Annual Conference of International Conference on Parallel Processing*, 1996

[12] TAU (Tuning and Analysis Utilities). See http://www.acl.lanl.gov/tau.

[13] H. Brunst and A. Malony and S. Shende and R. Bell, "Online Remote Trace Analysis of Parallel Applications on High-Performance Clusters," *International Symposium on High-Performance Computing*, October 2003, to be published.

[14] W.Nagel, A.Arnold, M.Weber, H.Hoppe, and K.Solchenbach, "Vampir: Visualization and Analysis of MPI Resources," *Supercomputing*, **12**(1):69–80, 1996.

[15] J. Vetter, "Computational Steering Annotated Bibliography," *SIGPLAN Notices*, **32**(6), pp. 40-44, 1997.